Impossible Differentials Automation: Model Generation and New Techniques

Emanuele Bellini[1]0000-0002-2349-0247, Paul Huynh[1]0000-0002-6965-3427, David Gerault[1]0000-0001-8583-0668, Andrea Visconti[2]0000-0001-5689-8575, Alessandro De Piccoli[2]0000-0002-6399-3164 and Simone Pelizzola[2]0009-0006-3991-1161

[1] Technology Innovation Institute, Abu Dhabi, UAE. {name.lastname}@tii.ae
[2] Università degli Studi di Milano, Milan, Italy, {name.lastname}@unimi.it

Abstract. In this paper, we aim to enhance and automate advanced techniques for impossible differential attacks. To demonstrate these advancements, we present improved attacks on the LBlock and HIGHT block ciphers. More precisely, we (a) introduce a methodology to automatically invert symmetric ciphers when represented as directed acyclic graphs, a fundamental step in the search for impossible differential trails and in key recovery techniques; (b) automate the search for impossible differential distinguishers, reproducing recent techniques and results; (c) present a new hybrid model combining cell-wise properties and bit-wise granularity; (d) integrate these techniques in the automated tool CLAASP; (e) demonstrate the effectiveness of the tool by reproducing a state-of-the-art 16-round impossible differential for LBlock previously obtained using a different technique and exhibiting a new 18-round improbable trail; (f) improve the state-of-the-art single-key recovery of HIGHT for 27 rounds, by automating the use of hash tables to current state-of-the-art results.

Keywords: Impossible differential; LBlock; HIGHT; CLAASP; Automated cryptanalysis

1 Introduction

Understanding the security properties of block ciphers is one of the most fundamental research areas in symmetric cryptanalysis, because, unlike public-key cryptosystems, their security cannot be convincingly proven based on a security assumption, but requires evaluating them against known attacks; on the other hand, secure block ciphers are a cornerstone for all modern applications and serve as building blocks for many primitives.

The most prominent techniques for block cipher cryptanalysis are derived from Differential cryptanalysis, introduced in the late 1980s by Biham and Shamir [BS90]. Differential attacks study the propagation of a bitwise XOR difference δ through the block cipher. When such a differential $\delta \to \gamma$ has a high probability, the attack is a classical differential attack. When it has probability 0, it is an Impossible Differential (ID) attack, as introduced independently by Biham et al. [BBS99] and Knudsen [Knu98].

Impossible differential cryptanalysis is an important part of the cryptographic landscape and was notably the first technique to break seven rounds of AES-128 [LDKK08]. To this day, the best-known cryptanalysis on CAMELIA [BNS14] is an impossible differential attack.

Finding differential distinguishers, and by extension, impossible differentials, has been a challenge for many years. Automated tools, mostly relying on CP, SAT, and MILP, have played a prominent role in making this task easier. However, the search for impossible differential remains more difficult than that of classical differentials. Intuitively, a classical differential distinguisher requires showing the existence of one differential trail. On the other hand, an impossible differential requires proving that such a property does not exist. Given the size of the search space, it is significantly easier to show that a property exists for some choice of input and output difference, than to prove that no trail connecting them exists.

The usual approach is to either enumerate a small subspace of the possible input and output differences and test the existence of a trail linking them with a MILP-based model, as in Sasaki and Todo's [ST17] and Cui et al. [CJF$^+$16], or use a miss-in-the-middle approach, where probability 1 differentials in the forward and backwards direction are shown to be incompatible, as in [SGWW20]. Improving these automated methods is an active research area in the field.

In addition to the search for distinguishers, recent works have focused on finding optimal attacks. In particular, it is well known that the longest or most likely distinguisher is not always the best for key recovery and that structural properties, such as the position of the active patterns, play an important part. Recognizing this, Hadipour et al. [HSE23] proposed a generic, satisfiability-based CP model to unify ID, ZC, and integral attacks, incorporating the complexity of the corresponding key recovery directly in their model. Building on this, the authors in [HGSE24] introduced a series of improvements. These include a flexible contradiction-location mechanism, a bit-wise model to address weakly aligned ciphers, and a CP model for the partial-sum technique. Their methodology demonstrated versatility by yielding significant improvements across diverse cipher designs, from strongly aligned ones like ForkSKINNY [ALP$^+$19] and QARMAv2 [ABD$^+$23] to weakly aligned ones such as Ascon [DEMS21] and PRESENT [BKL$^+$07]. Another work that recently sought to improve the key recovery procedure of impossible attacks was presented at Eurocrypt 2024 [BDD$^+$24]. In this paper, Boura et al. introduced an automatic tool to find the most efficient key-guessing order. Although limited to SPN ciphers with a linear or nearly linear key schedule, it represents progress toward fully automated attacks.

A broader trend in symmetric key cryptanalysis is to introduce such techniques into automatic tools, such as CLAASP [BGG$^+$23a], so that they can be easily generalized to other primitives without the need for extensive and error prone dedicated manual implementation. As a recent example, the NSA block cipher ARADI, published without a security analysis, was evaluated automatically against most common attacks a few weeks after its release [BFG$^+$24] using such a tool. For this reason, it is important to design cryptanalytic techniques that can be easily integrated into such automated frameworks.

This paper focuses on improving and automating state-of-the-art techniques for impossible differential attacks. These improvements are demonstrated by improving attacks on the LBlock [WZ11] and HIGHT [HSH$^+$06] block ciphers.

1.1 Our contributions

- We propose a fully automated search for impossible differential distinguishers, automating the techniques presented by Hadipour *et al.*'s framework [HGSE24] and introducing a novel technique to automatically generate the inverse of a symmetric cryptographic primitive when represented as a directed acyclic graph.
- We propose and automate a new *hybrid* model that combines the cell-wise properties of [SGWW20] and the granularity of the bit-wise models proposed in [CZZ22] and [HGSE24].
- In the *related-key* setting, our hybrid model also supports probabilistic transitions, which allow us to detect more incompatibilities than a standard truncated model, by leveraging the impossible differential clustering effect.

- We apply this model to LBlock [WZ11] and find 16-round impossible differentials, including some previously obtained using the different technique of [CJF+16]. We also show that the 17-round improbable trail presented in [CZZ22] is invalid and we exhibit an 18-round *improbable* trail that is valid for about $2^{-0.83}$ of the key space. Previous results only reached 16 rounds.

- We extend the open-source cryptanalysis library CLAASP with our models to make our results easy to reproduce and make the application of our techniques to more ciphers straightforward (as it only requires implementing the cipher in CLAASP and running the corresponding search function from the library);

- We extend the automatic technique of [HGSE24] for ID attacks with the use of hash-tables to reduce the time complexity as in [CWP12]. We apply this technique to the cryptanalysis of the HIGHT block cipher by improving the state-of-the-art 27-round *single-key* recovery attack, reaching a time complexity of $2^{118.9}$ and a data complexity of 2^{55} (previous time complexity was $2^{120.58}$ and data complexity $2^{59.3}$).

1.2 Outline

The paper is organized as follows. In Section 2, we introduce preliminary concepts about Impossible Differential Cryptanalysis (IDC) and describe automatic search tools applied to the search of impossible differentials. In Section 3, we depict a methodology to automatically invert a symmetric cipher when represented as a directed acyclic graph (a common representation used in automated tools). In Section 4 and Section 5, we focus on LBlock. Firstly, we describe previously impossible differential cryptanalysis results and disprove an existing result. Secondly, we develop a new hybrid approach, a unified model to identify impossible differentials using both bit-based and cell-based properties. We finally demonstrate its application by presenting a new 18-round trail. In Section 6 and Section 7, we focus on HIGHT. As done for LBlock, we initially present previous impossible differential results described in the literature. We then improve an existing model to find ID attacks, by exploiting the concept of hash tables, and show the improved results in the ID cryptanalysis of the cipher. In Section 8, we then draw conclusions.

2 Preliminaries

2.1 Differential and Impossible Differential Cryptanalysis

Differential cryptanalysis, introduced by Biham and Shamir in [BS91], studies the propagation of a difference through a cryptographic function. For a block cipher $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$, differential cryptanalysis is interested in differentials $\Delta_{in}, \Delta_{out}$ such that $E_k(x) \oplus E_k(x \oplus \Delta_{in} = \Delta_{out}$ holds with a high probability over all plaintexts x and keys k. The Differential Distribution Table (DDT) of a cryptographic list the probabilities of each differential transition.

Conversely, Impossible Differential Cryptanalysis (IDC) focuses on differentials with probability 0, *i.e.*, $\Delta_{in}, \Delta_{out}$ such that $E_k(x) \oplus E_k(x \oplus \Delta_{in} = \Delta_{out}$ has no solution.

The search for impossible differentials usually relies on a miss-in-the-middle approach, where two probability 1 differentials are incompatible. More specifically, let $\Delta_{in} \to \gamma$ be a probability 1 differential on r_f rounds in the forward direction, and let $\delta \leftarrow \Delta_{out}$ be a probability 1 differential on r_b rounds in the backward direction. If $\gamma \neq \delta$, then $\Delta_{in}, \Delta_{out}$ is an impossible differential on $r_f + r_b$ rounds.

2.2 Automatic Search Tools for Impossible Differential Trail Search

Optimization and operational research tools have become prominent in symmetric key cryptanalysis. Following the introduction of Mixed Integer Linear Programming (MILP) models for differential cryptanalysis [MWGP11], SAT and Constraint Programming (CP) have also been used for various cryptanalysis tasks. In these declarative frameworks, a model describing the problem in terms of *variables* and *constraints* is solved by a dedicated solver. The formalism in which the problems are described depends on the technique: boolean CNF formulas for SAT, linear inequalities for MILP, and generic constraints for CP.

Such automatic tools have been successfully applied to the search for impossible differentials in recent years. Two independent works by Sasaki and Todo [ST17] and Cui et al. [CJF+16] proposed the use of MILP to identify impossible differentials. Both approaches involve modeling the differential properties of the cipher and imposing constraints on the input and output differences until the problem becomes unsatisfiable. However, because this method requires fixing specific input and output differences, optimality is only confirmed through an exhaustive search over all possible input-output difference pairs. Sun et al. [SGWW20] first attempted to overcome this limitation with a CP-based method to find deterministic truncated differential trails in cell-oriented ciphers. Subsequent works expanded this approach to handling bit-oriented operations [CZZ22] and incorporated estimates of the overall attack complexity [HSE23, HGSE24].

2.2.1 Sun et al.[SGWW20]

Method The tool employs the *miss-in-the-middle* strategy to identify ID trails in cell-oriented ciphers. By converting the search for truncated differentials into a CSP, it separately enumerates forward truncated trails $\Delta_{in_0} \to \Delta_{out_0}$ and backward truncated trails $\Delta_{out_1} \leftarrow \Delta_{in_1}$. Incompatibility between the Δ_{out_0} and Δ_{out_1} is then verified outside the CP model.

Variables Consider $\Delta X = (\Delta X_0, \Delta X_1, \dots \Delta X_{l-1})$, the difference of the internal state X of size $l \cdot s$ bits, with $X_i \in \mathbb{F}_2^s$. For each cell difference ΔX_1, two variables are introduced: $\delta X_i \in \{0,1,2,3\}$ represents the differential pattern of ΔX_i and $\zeta X_i \in \{-2,-1,0,\dots,2^{s-1}\}$ represents the actual value of ΔX_i:

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_i = 0 \ (Z) \\ 1 & \text{if } \Delta X_i \text{ is nonzero and fixed } (N) \\ 2 & \text{if } \Delta X_i \text{ is nonzero } (N^*) \\ 3 & \text{if } \Delta X_i \text{ is unknown } (U) \end{cases} \qquad \zeta X_i \in \begin{cases} \{0\} & \text{if } \delta X_i = 0 \\ \{1,\dots,2^{s-1}\} & \text{if } \delta X_i = 1 \\ \{-1\} & \text{if } \delta X_i = 2 \\ \{-2\} & \text{if } \delta X_i = 3 \end{cases}$$

Constraints

- For the **branching** operation, the differential patterns satisfy:

$$\delta Y_0 = \delta X \text{ and } \zeta Y_0 = \zeta X \text{ and } \delta Y_1 = \delta X \text{ and } \zeta Y_1 = \zeta X$$

- For the **XOR** operation $Y = X_0 \oplus X_1$, the differential patterns satisfy:

```
if $\delta X_0 + \delta X_1 > 2$ then $\delta Y = 3$ and $\zeta Y = -2$
elseif $\delta X_0 + \delta X_1 = 1$ then $\delta Y = 1$ and $\zeta Y = \zeta X_0 + \zeta X_1$
elseif $\delta X_0 = \delta X_1 = 0$ then $\delta Y = 0$ and $\zeta Y = 0$
elseif $\zeta X_0 + \zeta X_1 < 0$ then $\delta Y = 2$ and $\zeta Y = -1$
elseif $\zeta X_0 = \zeta X_1$ then $\delta Y = 0$ and $\zeta Y = 0$
else $\delta Y = 1$ and $\zeta Y = \zeta X_0 \oplus \zeta X_1$ endif
```

- For the bijective **S-box** application $Y = S(X)$, the differential patterns satisfy:

$$\delta Y \neq 1 \text{ and } \delta X + \delta Y \in \{0, 3, 4, 6\} \text{ and } \delta Y \geq \delta X \text{ and } \delta Y - \delta X \leq 1$$

- For the $m \times m$ **MDS matrix** $Y = M(X)$, the differential patterns satisfy:

$$\begin{aligned}
&\text{if } \sum_{i=0}^{m-1} \delta X_i \equiv 0 \text{ then } \delta Y_0 = \delta Y_1 = \cdots = \delta Y_{m-1} = 0 \\
&\text{elseif } \sum_{i=0}^{m-1} \delta X_i \equiv 1 \text{ then } \delta Y_0 = \delta Y_1 = \cdots = \delta Y_{m-1} = 2 \\
&\text{elseif } \sum_{i=0}^{m-1} \delta X_i \equiv 2 \text{ and } \sum_{i=0}^{m-1} \zeta X_i < 0 \text{ then } \delta Y_0 = \delta Y_1 = \cdots = \delta Y_{m-1} = 2 \\
&\text{else } \delta Y_0 = \delta Y_1 = \cdots = \delta Y_{m-1} = 3 \text{ endif}
\end{aligned}$$

Limitations While the approach presents the benefit of automatically testing all input patterns, the S-box model does not take into account the DDT itself, limiting its precision. Furthermore, the forward and backward trail searches are conducted independently, rather than within a single model. Finally, the model is not applicable to ARX constructions.

2.2.2 Cao et al.[CZZ22]

An extension to ARX ciphers was introduced by Cao et al.[CZZ22]. Their model relies on *undisturbed differential bits*, a concept previously defined by Teczan[Tez14]. For an input difference Δ_{in}, an output bit Δ_{out_i} is said to be *undisturbed* if its value is deterministically fixed by Δ_{in}; such bits play a valuable role in identifying bit-level incompatibilities.

Method The miss-in-the-middle technique (as used in [SGWW20]), in which the propagations are modeled with MILP, is combined with undisturbed bits.

Variables Consider $\Delta X = (\Delta X_0, \Delta X_1, \ldots \Delta X_{n-1})$, the difference of the internal state X of size n bits. Each bit of the state is associated with a variable $\delta X_i \in \{0, 1, 2\}$ that represents the value of ΔX_i:

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_i = 0 \\ 1 & \text{if } \Delta X_i = 1 \\ 2 & \text{if } \Delta X_i \text{ is unknown} \end{cases}$$

Constraints

- For the **XOR** operation $Y = X_0 \oplus X_1$, the differential patterns satisfy:

$$\begin{aligned}
&\text{if } \delta X_0 = 2 \text{ or } \delta X_1 = 2 \text{ then } \delta Y = 2 \\
&\text{else } \delta Y = \delta X_0 \oplus \delta X_1 \text{ endif}
\end{aligned}$$

- For the **AND** operation $Y = X_0 \wedge X_1$, the differential patterns satisfy:

$$\begin{aligned}
&\text{if } \delta X_0 + \delta X_1 = 0 \text{ then } \delta Y = 2 \\
&\text{else } \delta Y = 2 \text{ endif}
\end{aligned}$$

- For the **Modular Addition** operation $Z = X \boxplus Y$, where $X = (x_{n-1}, \dots, x_0)$, $Y = (y_{n-1}, \dots, y_0)$ and $Z = (z_{n-1}, \dots, z_0)$, the differential patterns satisfy:

$$\begin{aligned}
&\delta z_0 = \delta x_0 \oplus \delta y_0,\ c_0 = f_1(\delta x_0, \delta y_0)\\
&\delta z_1 = \delta x_1 \oplus \delta y_1 \oplus c_1,\ c_1 = f_2(\delta x_1, \delta y_1, c_0)\\
&\vdots\\
&\delta z_{n-2} = \delta x_{n-2} \oplus \delta y_{n-2},\ c_{n-1} = f_2(\delta x_{n-2}, \delta y_{n-2}, c_{n-3})\\
&\delta z_{n-1} = \delta x_{n-1} \oplus \delta y_{n-1} \oplus c_{n-2}
\end{aligned}$$

$$\text{where } f_1(x,y) = \begin{cases} 0 & \text{if } x+y=0 \\ 2 & \text{otherwise} \end{cases}, \quad f_2(x,y,c) = \begin{cases} 0 & \text{if } x+y+c=0 \\ 2 & \text{otherwise.} \end{cases}$$

- For the **S-box** application $Y = S(X)$, where $X = (x_{n-1}, \dots, x_0)$, and $Y = (y_{m-1}, \dots, y_0)$, the DDT restricted to undisturbed bits must be encoded. For each input difference ΔX, let us consider the set $\mathcal{P}_{\Delta X}$ of undisturbed bits positions of S under ΔX. For p in $\mathcal{P}_{\Delta X}$, we denote by b_p the undisturbed bit value of at position p of the output, that is, $\Delta y_p = b_p$. If $\mathcal{P}_{\Delta X} \neq \emptyset$ then $\delta X = (\delta X_{n-1}, \dots, \delta X_0)$ propagates to the output difference pattern $\delta Y = (\delta Y_{m-1}, \dots, \delta Y_0)$, where

$$\delta Y_i = \begin{cases} b_i & \text{if } i \in \mathcal{P}_{\Delta X} \\ 2 & \text{otherwise.} \end{cases}$$

  In all the other cases, the output is all unknown: $(\delta Y_{m-1}, \dots, \delta Y_0) = (2)_{i=0}^{m-1}$.

Limitations Despite its applicability to bit-oriented primitives, the approach has several shortcomings. In this model, if a nonzero input does not produce any undisturbed bits, the output is assigned an unknown value. For bijective operations, this results in the loss of crucial information, as the output is guaranteed to be nonzero. Furthermore, the method requires fixing the input patterns, which represents a regression compared to the approach of [SGWW20]. Finally, some results in the paper appear to be incorrect, indicating potential issues in the model or its implementation; these are discussed in Subsection 4.1.

2.2.3 Hadipour et al.[HSE23]

Method `Zero` is a tool developed by Hadipour et al. for identifying impossible differential, zero-correlation and integral attacks on block ciphers. Like the approach in [SGWW20], it formulates the differential propagations as a constraint optimization problem. However, in this case, the forward and backward trail searches are integrated into a single unified model. The encoding is the one employed in [SGWW20] and thus we refer to Subsubsection 2.2.1 for the detailed definitions of variables and constraints. A key strength of this approach is that the tool additionally gives the time complexity of the corresponding key recovery, allowing to find full impossible differential attacks.

Limitations The approach is not applicable to ARX-based constructions, as the encoding of the model only considers cell-wise patterns. Another drawback is that the number of rounds forward and backward must be specified in advance.

2.2.4 Hadipour et al.[HGSE24]

Earlier this year, Hadipour et al. released `Zeroplus`, an improvement of `Zero` [HSE23]. This updated version extends the tool to weakly aligned primitives by including bit-wise propagations for the branching, XOR and S-box operations, with the inclusion of undisturbed bits, following the approach of [CZZ22]. While `Zeroplus` still does not apply to ARX, it addresses the second limitation of the previous version of the tool, by automating the identification of the middle round, removing the need for manual specification.

Going Further All the previously described approaches were implemented manually for specific ciphers. This process is often tedious and error-prone, and work towards more automation is important within the community. We propose a method to automatically generate the decryption function of a cipher within an automated tool, as a stepping stone towards the fully automatic generation of impossible differential models.

3 Automating Cipher Inversion

As noted in the previous section, automation can help limit the risk of human oversight. With this in mind, we aimed to push this principle further by eliminating the need to manually construct the corresponding impossible differential model for each cryptographic primitive. This approach aligns with the philosophy behind automated cryptanalysis tools such as CLAASP [BGG+23a], which served as a starting point for our efforts. Indeed, CLAASP already supports the generation of SAT, SMT, CP, and MILP models for various attack scenarios. In particular, the tool can automate the search of differential and linear distinguishers. The first challenge was the systematic inversion of a cipher from its representation in CLAASP to model the backward propagations. We also notice that inversion is a fundamental step in many key recovery techniques. In CLAASP, a primitive is described as a list of connected components, forming a directed acyclic graph. Each component is a dictionary containing an identifier, a type, and a description specifying the operation. It also includes the identifiers of its input components, their bit positions, and the input and output sizes.

3.1 Method description

To ensure generality, the inversion process operates sequentially on individual components, starting from the output and working backward through the cipher to the input, rather than processing groups of components.

The first step is to define how each type of component should be handled. For components representing a *bijective operation*, the inversion simply involves swapping the input and output and applying the inverse permutation. The condition for this component to be *inverted* successfully is that all output bits must be available by the time it is reached. *Non-bijective operations*, however, require more careful consideration. For instance, the shift operation is not invertible. If this component is encountered during the inversion process, it needs to be *evaluated*; in other words, its input bits must be available for the inversion to succeed. A practical illustration would be a 2-branch Feistel structure where the Feistel function is a shift operation. In this case, inversion is still possible because the input to the Feistel function is also available via the round output.

Another type of operation to examine more thoroughly is the XOR. Given $a \oplus b = c$, the operation can be evaluated if a and b are known. Alternatively, it can also be inverted in two ways: either $a = b \oplus c$ or $b = a \oplus c$, depending on which input is available first.

Given a target cipher C, the inversion process begins by establishing a list L of available components. Specifically, at any stage of the process, a component is considered available if it can either be evaluated or inverted, given the cipher output and the components previously made available up to that point. Initially, L only contains the output of the primitive to be inverted as it becomes an input to the inverse: $L =$ [`cipher_output`]. All the other components remain in a separate list T, representing components yet to be inverted. As the components are processed, they are moved from T to L. The inversion is complete once T is empty. The procedure is detailed in algorithm 1. The `can_be_evaluated()` and `can_be_inverted()` functions are routines that verify whether enough input and/or output bits are available to process the component c, by establishing its connection to the elements of L. The functions `evaluate_component()` and `invert_component()` create a new component c' from c, based on the rule established for the operation it represents–as discussed above–and its connection to the elements of L. We illustrate this method on a toy example in Appendix C.

Algorithm 1: Cipher inversion algorithm

```
Input   : Target cipher $C$
Output : Inverse cipher $C_{inv}$
$L$ = [cipher_output]
$T = [c \mid c \in C]$,
while $|T| > 0$ do
    for $c$ in $L$ do
        if can_be_evaluated($c, L$) = True then
            $c'$ = evaluate_component(c, $L$)
            $L = L + [c']$
            remove $c$ from $T$
        end
        else if can_be_inverted(c, $L$) = True then
            $c'$ =invert_component(c, $L$)
            $L = L + [c']$
            remove $c$ from $T$
        end
    end
end
Build $C_{inv}$ from the components list $L$
```

3.2 Limitations of this approach

This inversion method relies exclusively on local information and processes each component independently, rather than considering larger groups of operations. Consequently, certain components are challenging to invert. For instance, in the linear layer of Ascon [DEMS21], each row x is updated through a series of shifts and XORs. Specifically, x is shifted by two values, r_0 and r_1, and these shifted versions are then combined with the original x through an XOR: $y = x \oplus (x \ggg r_0) \oplus (x \ggg r_1)$. From this expression, the original value x can be recovered from y by solving the equation. However, consider an implementation where this transformation is broken into simpler components, as follows:

$$u = x \ggg r_0;\; v = x \ggg r_1;\; y = x \oplus u \oplus v.$$

In such a case, the component producing y is simply viewed as a 3-input XOR operation with no discernible connection to the earlier shifts. This lack of global context prevents the inversion process from reconstructing x. On the other hand, the entire linear layer of Ascon can be expressed as a binary matrix multiplication, which is inherently easy to invert. As such, the effectiveness of this inversion method heavily depends on how the cipher is implemented.

Table 1: Previous results of impossible differential distinguishers for LBlock

Setting	Rounds	Key Space Covered	Search Method	Ref.
Single-Key	14	Full	$\mathcal{U}$-method	[WZ11]
Related-Key	15	Full	Dedicated	[MN12]
Related-Key	16	Full	Dedicated	[WWZ14]
Related-Key	16	Full	MILP (Infeasibility-based)	[CJF+16]
Related-Key	17	2^{-2}	MILP (Deterministic Truncated + Undisturbed Bits)	[CZZ22]

4 Previous ID cryptanalysis results on LBlock

LBlock is a lightweight block cipher proposed by Wu and Zhang at ACNS 2011 [WZ11]. Its description is given in Appendix A. In the design document, the authors provided an impossible differential analysis in the single-key setting covering 20 rounds by using a 14-round distinguisher found with Kim et al.'s $\mathcal{U}$-method [KHS+03]. Using the same characteristic, this result was improved by Karakoç et al. [KDH12], reaching 21 rounds. A 22-round attack was also devised, using a different 14-round impossible differential path. A similar characteristic was used by Boura et al. to break 23 rounds [BMNS14].

The first related-key analysis was performed by Minier and Naya-Plasencia [MN12]. Their analysis covered 22 rounds and took advantage of the low diffusion of the key schedule to find a 15-round impossible differential path. The weakness of the key schedule was once again used by Wen et al. [WWZ14], who proposed a dedicated algorithm to search for impossible differential trails in LBlock, allowing them to reach 16 rounds. In both cases, all the possible key values are split into a partition of differential trails. If a differential appears to be impossible for each trail, then it holds for the entire key space. In [CJF+16], Cui et al. reported several 16-round impossible differential paths. To this date, it is the highest number of rounds for which impossible differentials exist for the entire key space. In their approach, constraints on the input and the output of the cipher are added to a MILP model describing the differential behavior of the cipher. When the system reaches infeasibility, an impossible differential is found. For LBlock, the authors restricted their search to cases in which there is exactly one bit of difference between the two master keys and, at most, 1 bit of difference in the input and the output. In 2022, Cao et al. [CZZ22] proposed an automatic search tool that takes into account the so-called *undisturbed differential bits* already defined by Teczan [Tez14]. Additionally, they provided a 17-round impossible path in the weak related-key model found with their tool. All the main results on LBlock are summarized in Table 1.

4.1 Disproving the previous result on LBlock

In the work by Cao et al. [CZZ22], the authors present a 17-round related-key *improbable* differential trail for LBlock. Rather than modeling the key schedule with deterministic propagation patterns, they employ probabilistic differential propagations, shown in Table 2. This approach leads to distinguishers that are longer but are not valid across the entire key space. However, it appears that their implementation is flawed as the described key schedule trail follows invalid transitions.

More precisely, producing a subkey difference equal to `0x01800000` at round 5 is not possible if the subkey difference at round 2 equals `0x00030000`. Indeed, it is quite easy to observe that given any subkey $sk_i = (sk^i_{31} sk^i_{30} \dots sk^i_1, sk^i_0)$ of a round $i < 9$, the following equalities hold for the subkey of round $i + 3$[1]:

$$sk^{i+3}_{27\sim 24} = S_8(sk^i_{20\sim 17}). \quad (1)$$

[1]This was also observed in [WWZ14]

Table 2: Subkeys used in the 17-round improbable trail by [CZZ22]. $\alpha \in \{0,1,2,3\}, \beta \in 0,4,8,12, \gamma \in \{0,1\}, \delta \in \{0,2,4,6,8,10,12,14\}$.

Round	Subkey	Round	Subkey	Round	Subkey	Round	Subkey
0	00000000	5	00000000	10	00000000	15	00?00000
1	00030000	6	00000006	11	00000000	16	00000000
2	00000000	7	?0000000	12	$000\gamma\delta000$		
3	00000000	8	00000000	13	00000000		
4	01800000	9	$00000\alpha\beta0$	14	00000000		

For $\Delta sk_2 =$ `0x00030000` and $\Delta sk_5 =$ `0x01800000`, Equation 1 would imply that `0x1` $\rightarrow$ `0x1` is a valid transition for S_8, which is false. It is clear how crafting models manually can be an error-prone process, not as with fully automatic modeling.

5 New 18-round improbable distinguisher for LBlock

Upon the realization that the trail provided by [CZZ22] was incorrect, the natural approach was to search for other trails using a similar bit-wise model. One first observation is that the bit-based model with undisturbed bits presents one limitation, as already mentioned in Subsubsection 2.2.2: it is unable to keep track of groups of bits that are undetermined yet nonzero. This typically happens when a bijective operation maps a nonzero input difference to a truncated output difference with no undisturbed bit. Let us consider the first S-box of LBlock as an example:

$$S_0 = (14, 9, 15, 0, 13, 4, 10, 11, 1, 2, 8, 3, 7, 6, 12, 5).$$

There are 6 differential transitions with undisturbed bits: $(0000 \xrightarrow{S_0} 0000), (0001 \xrightarrow{S_0} ???1), (0010 \xrightarrow{S_0} ???1), (0011 \xrightarrow{S_0} 1??0), (1000 \xrightarrow{S_0} 1???), (1011 \xrightarrow{S_0} 0???)$.

In the bit-based model, all other cases are mapped to a fully undetermined output $\delta_{out} =????$, which provides no meaningful information for detecting incompatible cases. However, since the input is nonzero, the output difference should also be nonzero with probability 1, but this information is lost under this model. In contrast, the cell-wise model can track such propagations (we refer to description of the S-Box constraint in Subsubsection 2.2.1) and has proven to be effective for LBlock, as evidenced by the previous word-based impossible differential cryptanalysis results [WZ11, MN12, WWZ14].

Indeed, it aligns well with the round function, which involves *cell-aligned* 4-bit operations. However, the key schedule includes a rotation of 29 bits to the left and would thus benefit from a bit-based model. Additionally, the undisturbed bits of an S-box provide extra information that can lead to the detection of more impossible trails. Another case where the bit-wise model may provide more information than its cell-wise counterpart is when looking at an XOR operation between one fully known input and one partially known input: given $a =???1$ and $b = 0010$, the bit-wise representation is $a \oplus b =???1$ while the cell-wise model abstracts this operation as $N \oplus N^* = U$.

These observations motivated us to investigate a *hybrid model*, which integrates bit-based and cell-based representations into a unified framework. This approach eliminates the need to choose between the two models while retaining their respective advantages.

5.1 Hybrid model for the distinguisher search

The core idea of the model is to capture both notions of undisturbed bits and of active groups of bits in S-box outputs and, more generally, in any nonlinear operation that is bijective. As such, it can be seen as an extension of the bit-wise deterministic truncated model with undisturbed bits, to which cell-wise properties are added.

5.1.1 A toy example

Let us look at a toy example to showcase our hybrid model. We consider a simple Feistel cipher acting on an 8-bit state $(x_l||x_r)$, where the Feistel function F is defined by a round key addition followed by the application of an S-box: $F(x_l, k_i) = S_0(x_l \oplus k_i)$, where S_0 is the first S-box used in LBlock, previously defined.

For the sake of simplicity, the key schedule is linear and produces the 4-bit round keys differences 1000, 0110, 1101, and 0101 at rounds 0, 1, 2, and 3, respectively. Fixing the input and output differences to (0000, 0000) and (0101, 0000), respectively, Figure 1 shows that the regular cell-based and bit-based models do not detect any incompatibility whereas the hybrid model finds the differential to be impossible. Indeed, one extra information that the hybrid model gains over the other models is that the right branch at the end of round 1 is undetermined but nonzero. More precisely, these models differ in their characterization of the result of $S(S(0000 \oplus 1000) \oplus 0110)$:

- for the cell-based model, $S(0000 \oplus 1000)$ is the result of an S-box evaluation on two fixed inputs, so it is an undetermined nonzero value α. $S(S(0000 \oplus 1000) \oplus 0110) = S(\alpha \oplus 0110)$ is then simplified to the output of an S-box evaluated on an unknown input[2], so the result is unknown;
- for the bit-based model, $S(0000 \oplus 1000) = 1???$ using the undisturbed bits of S, and thus $S(S(0000 \oplus 1000) \oplus 0110) = S(1??? \oplus 0110) = S(1???)$. Since 1??? has no undisturbed bits, the result is unknown;
- for the hybrid-based model, as done with the bit-based model, we have $S(0000 \oplus 1000) = 1???$ using the undisturbed bits of S and thus $S(S(0000 \oplus 1000) \oplus 0110) = S(1??? \oplus 0110) = S(1???)$. This is a bijective S-box evaluated on a nonzero input, so the result is undetermined but nonzero.

5.1.2 Model description

The hybrid model is an extension of the bit-wise model described in Subsubsection 2.2.2.

Data representation We recall that in bit-based approaches, for each round r, each bit difference is associated to an integer variable $\delta X_{i,r} \in \{0, 1, 2\}$, where 2 stands for the unknown bit. In the hybrid model, this domain is extended by introducing a unique integer $\mathtt{id}_{s,r}$ for each n-bit S-box s of round r that serves as an identifier. The value should not intersect with the base domain of the bit-wise representation, meaning that for all s and r, $\mathtt{id}_{s,r} > 2$.

$$\delta X_i = \begin{cases} 0 & \text{if } \Delta X_{i,r} = 0 \\ 1 & \text{if } \Delta X_{i,r} = 1 \\ 2 & \text{if } \Delta X_{i,r} \text{ is unknown} \\ \mathtt{id}_{s,r'} & \text{if } \Delta X_{i,r} \text{ is produced by S-box } s \text{ of round } r', \\ & \text{evaluated on a nonzero input difference} \end{cases}$$

Whenever a nonzero input difference has no undisturbed bits, the n bits of output are all set to $\mathtt{id}_{i,r}$ to indicate that even though the output is unknown, the sum of the bits is nonzero. Denoting n_{sb} the number of S-box per round and n_r the number of rounds, the new domain of each variable $\delta X_{r,i}$ thus becomes $\mathcal{D}(\delta X_{ri}) = \{0, 1, 2\} \bigcup_{s \in \mathcal{N}_{sb}, k \in \mathcal{N}_r} \{\mathtt{id}_{s,k}\}$, where $\mathcal{N}_{sb} = \{0, \ldots, n_{sb} - 1\}$ and $\mathcal{N}_r = \{0, \ldots, n_r - 1\}$. The modeling of the S-box and XOR operations needs to be changed to reflect this domain extension.

[2]In our cell-based model, the XOR of an undetermined nonzero input with a fixed input is unknown.

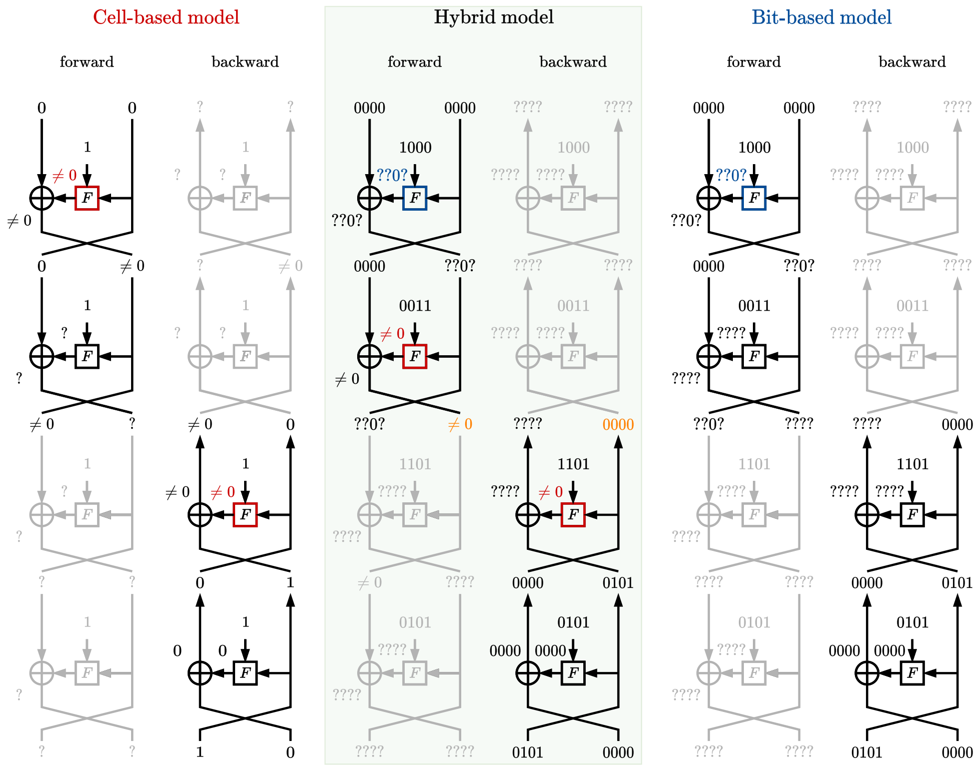

Figure 1: Comparison of the cell-based, bit-based, and hybrid models.

Extended modeling of a bijective S-box For the S-box application $Y = S(X)$, where S is the s-th S-box at round r, $X = (x_{n-1}, \ldots, x_0)$, and $Y = (y_{n-1}, \ldots, y_0)$, the encoding still uses information from the DDT restricted to undisturbed bits. For this, we introduce new variables $\beta X_{i,r} \in \{0, 1, 2\}$ for each bit difference ΔX_i. These variables satisfy the bit-wise constraints of the S-box. Consider the sets $\mathcal{P}_{\Delta X}$ and $\{b_p \mid p \in \mathcal{P}\}$ as defined in Subsubsection 2.2.2. If $\mathcal{P}_{\Delta X}$ is not empty then $\beta X = (\beta X_{n-1}, \ldots, \beta X_0)$ propagates to the output difference $\beta Y = (\beta Y_{n-1}, \ldots, \beta Y_0)$, where

$$\beta Y_{i,r} = \begin{cases} b_i & \text{if } i \in \mathcal{P}_{\Delta X} \\ 2 & \text{otherwise.} \end{cases}$$

In all the other cases, $(\beta Y_{m-1}, \ldots, \beta_{Y_0}) = (2)_{i=0}^{n-1}$. In turn, extended output patterns $\delta Y = (\delta Y_{n-1}, \ldots, \delta Y_0)$ satisfy

$$\delta Y = \begin{cases} \beta_Y \text{ or } \mathtt{id}_{s,r} & \text{if } \exists i \in \{0, \ldots, n-1\},\ \delta X_i = 1 \\ \mathtt{id}_{s,r} & \text{if } \forall i \in \{0, \ldots, n-1\},\ \delta X_i = \mathtt{id}_{s',r'} \\ 2 & \text{otherwise.} \end{cases}$$

Extended modeling of the XOR For the XOR operation $Y = X_0 \oplus X_1$, the propagation of the differential pattern follows:

$$\begin{aligned} &\text{if } \delta X_0 < 2 \wedge \delta X_1 < 2 \text{ then } \delta Y = \delta X_0 + \delta X_1 \\ &\text{elseif } \delta X_0 > 2 \wedge \delta X_1 = 0 \text{ then } \delta Y = \delta X_0 \\ &\text{elseif } \delta X_0 = 0 \wedge \delta X_1 > 2 \text{ then } \delta Y = \delta X_1 \\ &\text{else } \delta Y = 2 \end{aligned}$$

Table 3: New 16-round impossible differentials found with our model

Δin	Δout	ΔK
$0x0$	$0x0$	$0xb$
$0x0$	$0x0$	$0x580$
$0x60000000000$	$0x0$	$0x580$

Objective function With this extended approach, two types of incompatibilities can be detected. Denoting $\delta Xu_{r,i}$ (respectively $\delta Xl_{r,i}$) the difference in bit i of the internal state at round r in the forward (respectively backward) direction:

1. a **bit-based contradiction** occurs if there exist a round r and a bit position i for which $\delta Xu_{r,i} + \delta Xl_{r,i} = 1$;

2. a **word-based contradiction** occurs if there exist a round r, n bit positions $i_0, \ldots i_{n-1}$ and an S-box index s that verify:

$$\forall j \in \{i_0, \ldots, i_{n-1}\}, (\delta Xu_{r,i_j} = \mathtt{id}_{s,r} \wedge \delta Xl_{r,i_j} = 0).$$

5.2 A first application to LBlock

The longest impossible differentials covering the full key space of LBlock were identified by Cui et al. [CJF+16], who presented several 16-round impossible differentials in the related-key setting. These results cannot be reproduced using the bit-based model due to the limitations discussed earlier in this section. In Appendix D, Table 7 and Table 8 detail the propagation in the subkeys and in the state under this model, for the differential $0 \xrightarrow{16r} 0$, with a single active bit located at k_{11} in the master key.

Using the hybrid model, we recover this differential. The resulting state is shown in Table 9 of Appendix D. We also provide a script that can find other 16-round impossible differentials that were not listed in Cui et al.'s paper (Listing 2) and reported them in Table 3. This highlights the hybrid model's advantage in bridging gaps left by the bit-based approach. However, it is still unable to capture all the other differentials reported by Cui et al. due to differences in our approach compared to theirs.

The strategy described in [CJF+16] involves testing the feasibility of a differential model given specific constraints on the input and output; if no solution exists, the differential is deemed impossible. One limitation of this model is that the input and output differences must be fixed in advance but it offers one advantage over ours: if the set of all *differential characteristics* composing a differential can be partitioned into subsets $P = \cup_{p_i}$ such that, for each pair (p_i, p_j), the cell or bit positions of the incompatibility differ, Cui et al.'s model will successfully detect this differential as impossible. In contrast, our truncated deterministic approach will fail to do so.

One example among the differentials found by Cui et al. is

$$(\mathtt{0x00000000}, \mathtt{0x00000000}) \xrightarrow[\Delta K = \mathtt{0x40}]{16r} (\mathtt{0x00000000}, \mathtt{0x00000000}).$$

An examination of the propagations in the key schedule (depicted in Table 10) reveals that the location of the contradiction at round 9 depends on the most significant bit of the first byte of the subkey produced at round 8:

- if it equals 1, the differential is impossible due to an inconsistency at byte 15,

- otherwise, the differential is impossible due to an inconsistency at byte 11.

Table 4: 16-round impossible trails found by [CJF+16] tested against our model.

Δin	Δout	ΔK	SAT differential model	our model
0	0 or $1 \ll j$, $j \in \{40, 41, 42, 43\}$	$1 \ll 11$	✓	only $\Delta_{out} = 0$ or $1 \ll j$, $j \in \{43, 42\}$
0	0 or $1 \ll j$, $j \in \{52, 53, 54, 55\}$	$1 \ll 10$	✓	no
0	0	$1 \ll 6$	✓	✓
0	0 or $1 \ll j$, $j \in \{44, 45, 46, 47\}$	4	✓	no
0	0	2	no	no ($p = 2^{-0.093}$)
0	0	1	no	no ($p = 2^{-0.093}$)

5.3 A new result using probabilistic trails

As observed in Subsection 5.2, for LBlock, given fixed plaintext, ciphertext, and key differences, it is possible for the incompatibility to occur at different locations, depending on the S-box outputs of the key schedule. Subkeys partitioning has previously been used to manually find impossible differentials [MN12, WWZ14]. Thus, allowing probabilistic differential transitions for the key schedule in our model can lead to the identification of more impossible differentials.

The modeling of the two key schedule S-boxes S_8 and S_9 were changed to encode their standard DDT and the corresponding probability, as done for automated differential cryptanalysis [GMS16, AST+17]. Then, we used this probabilistic hybrid model to enumerate all trails of weight less than 20. Indeed, each transition of S_8 and S_9 has a probability of at least 2^{-3}, and with a small number of active bits in the key, we can expect to have very few active S-boxes even for a high number of rounds. The trails found are then merged to obtain differentials.

Application to 16 rounds of LBlock For 16 rounds, we are able to retrieve more of the 16-round impossible differentials found by the authors of [CJF+16]. The results are summarized in Table 4. While some missing instances are due to a limitation of our approach, it should be noted we were unable to reproduce the paper's results when either bit k_0 or k_1 of the master key is active, even with the strategy described by the authors. In fact, the SAT differential model generated by CLAASP shows that the differentials are satisfiable.

Application to 18 rounds of LBlock For 18 rounds, among the 121 trails of weight less than 20, 6 are part of the following differential

$$(\texttt{0x00000000}, \texttt{0x00000008}) \xrightarrow[\Delta K = \texttt{0x40000000}]{18r} (\texttt{0x00000000}, \texttt{0x00000000}).$$

This impossible differential holds on average for 2^{45} keys and requires a subkey at round 10 of the form $\texttt{0x}\alpha\texttt{0000000}$, where $\alpha = \texttt{0x0}$ or $\texttt{0x8}$, which occurs with probability $2^{-0.83}$. This new trail is reported in Table 11 and the patterns for the subkeys are given in Table 12 of Appendix E. Additionally, verification scripts based on CLAASP are provided in Listing 3 and Listing 4 of Appendix E.

6 Previous ID cryptanalysis results on HIGHT

HIGHT is a lightweight block cipher proposed by Hong et al. at CHES 2006 [HSH+06]. Its description is provided in Appendix B. The security analysis performed by the authors showed impossible differential results up to 18 rounds. Later works brought that number up to 27 rounds for single-key impossible differential cryptanalysis and 31 rounds for the related-key setting. In [Lu07], Lu presented new impossible differentials reaching 25 rounds (rounds 6-30) in the single-key scenario and 28 rounds (rounds 3-30) in related-key, while removing the initial transformation of the cipher. Ozen et al. extended the attack

Table 5: Previous results of impossible differential cryptanalysis of HIGHT

Scenario	**Rounds**	**Transformations**	**Time**	**Data**	**Memory**	Ref.
Single-key	18 (1-18)	Both	$2^{109.2}$ enc.	$2^{46.8}$ plaintexts	/	[HSH+06]
Single-key	25 (6-30)	Only final	$2^{126.75}$ enc.	2^{60} plaintexts	/	[Lu07]
Single-key	26 (1-26)	Only final	$2^{119.53}$ enc.	2^{61} plaintexts	2^{109} B	[ÖVTK09]
Single-key	27 (4-30)	Both	$2^{126.6}$ enc.	2^{58} plaintexts	2^{120} B	[CWP12]
Single-key	27 (4-30)	Both	$2^{120.58}$ enc.	$2^{59.3}$ plaintexts	$2^{107.4}$ B	[AAA+18]
Related-key	28 (3-30)	Only final	$2^{125.99}$ enc.	2^{59} plaintexts	/	[Lu07]
Related-key	31 (1-31)	Only final	$2^{127.28}$ enc.	2^{64} plaintexts	2^{117} B	[ÖVTK09]
Related-key	32	Both	$2^{127.276}$ enc.	2^{64} plaintexts	2^{104} B	[RCT13]

to 26 (rounds 1-26) and 31 rounds (rounds 1-31) respectively, again without the initial transformation. In [CWP12] Chen et al. finally included the initial transformation, and attacked 27 rounds (rounds 4-30) in the single-key scenario, using hash tables to optimize the attack. This approach was refined by Azimi et al. in [AAA+18]. In [RCT13], Rostami et al. show a full-round attack for a class of weak keys in the related-key setting. A summary of all the results can be found in Table 5.

7 New results on HIGHT

The general steps for a key recovery attack, given an impossible differential, are as follows.

1. **Distinguisher extension**: The attacker propagates the impossible differential trail to the plaintext and ciphertext. The propagation is performed in the backward direction from the start of the differential to the plaintext and in the forward direction from the output difference to the ciphertext.

2. **Pairs generation**: In the pairs generation phase, pairs satisfying the extended differential are generated. This is usually done by generating some structures of 2 sets of plaintexts (resp. ciphertexts), where fixed difference bits are constant, and free difference bits vary so that all plaintexts (resp. ciphertexts) pairs in the structure satisfy the expected difference. The structures are then encrypted (resp. decrypted), and the ones satisfying the output (resp. input) extended difference are kept.

3. **Pairs elimination**: The pairs satisfying each round's difference are propagated deterministically forward from the plaintext and backward from the ciphertext, exhaustively guessing the involved key bits. Since the starting states will contain more unknown difference bits than in the successive states, this process leads to the elimination of some of the initial pairs, namely the ones not satisfying at least one round difference.

4. **Subkeys elimination**: When the last round of the extension of the distinguisher is reached, the subkey elimination step begins. Each remaining pair leads to the elimination of all the guessed subkeys which makes it satisfy the impossible differential. Indeed, if one of such subkeys were correct, then the impossible differential would be satisfied for at least one pair, leading to a contradiction.

5. **Exhaustive search** The remaining keys are exhaustively checked until the correct one is found.

Chen et al. introduced a variation of the attack in [CWP12] that uses hash tables to reduce the time complexity of the pairs elimination step, which is the heaviest one. In particular, portions of the distinguisher's extension are precomputed, in order to efficiently deduce the subkeys leading to the impossible differential when the guessed portion is reached during the pairs elimination phase. Therefore, instead of multiplying the complexities of each step, it is enough to add the sub-extension complexities of the evaluations. The hash table complexity gains are usually computed on top of an existing impossible differential, even though different impossible differentials may be better suited; in the following section, we account for this precomputation step in an impossible differential search model.

7.1 Modeling the search for the distinguisher and its extensions

Our model extends the approach of the bit-based version of [HGSE24] in two ways: first, we obtain a finer-grained complexity analysis by including the hash tables precomputation [CWP12] in the objective function. In addition, we let the solver choose the number of rounds of each part of the trail.

HIGHT operates on words of 8 bits using the XOR, modular addition, and left rotation. To model these operations, we use the bit-based encoding of [CZZ22] (Subsubsection 2.2.2). We recall that each bit difference is represented by a variable with values in the set $\{0, 1, 2\}$, where 0 and 1 correspond to known bit differences of 0 and 1, and 2 represents an unknown bit difference.

We use the high-level language MiniZinc [NSB+07] to create this model. In Appendix F, we provide the MiniZinc implementations of the different functions (Listing 5, Listing 6, Listing 7). In our standard model (without the use of hash tables), we use the following matrices:

```
array[0..7, 0..7] of var 0..2: P; %Plaintext difference
array[0..7, 0..7] of var 0..2: C; %Ciphertext difference
array[0..rounds, 0..7, 0..7] of var 0..2: St; %Round States differences
array[0..rounds, 0..7, 0..7] of var 0..2: Inv_St; %Round states differences evaluated
    from P and C
array[0..rounds, 0..7] of var 0..8: Fil; %Filtering bytes positions and dimension
array[0..rounds, 0..7] of var 0..1: Fil_N; %Ancestors bytes of the Filtering ones (
    bytes needed for their evaluation)
array[0..15] of var 0..15: KP; %Permutation of the master key bytes for guessing order
array[0..15] of var 1..16: KP_id; %Inverse of the permutation of the master key bytes
    for guessing order
array[0..rounds, 0..7] of var 0..16: MK; %Maximum index of msater key byte involved in
    byte evaluation
array[0..15] of var 0..128: P_El; %Number of eliminated pairs after each key byte guess
```

We denote the deterministic propagation of a state through a HIGHT round in the forward (resp. backward) direction by DirProp (resp. InvProp). We then have:

$$\begin{aligned}
\texttt{St}[i] &= \texttt{InvProp}(\texttt{St}[i+1]),\ \forall i \in \{0, \dots, n_I - 1\}\\
\texttt{St}[i] &= \texttt{DirProp}(\texttt{St}[i-1]),\ \forall i \in \{n_I + 1, \dots, n_M\}\\
\texttt{St}[i] &= \texttt{InvProp}(\texttt{St}[i+1]),\ \forall i \in \{n_M, \dots, n_F - 1\}\\
\texttt{St}[i] &= \texttt{DirProp}(\texttt{St}[i-1]),\ \forall i \in \{n_F + 1, \dots, n_r\}
\end{aligned}$$

where n_I is the end round of the initial extension, n_M is the meeting round of the impossible differential, and n_F is the start round of the final extension. $\texttt{St}[n_I]$ is the start state of the impossible differential and $\texttt{St}[n_F]$ is the end state of the impossible differential, which in our model are all variables. For a differential to be impossible, there must be an inconsistency in $\texttt{St}[n_M]$, encoded through an additional row in the state table, where both states are checked for incompatibility.

Modeling the pairs elimination phase In this paragraph, we model the set of all the bytes to be guessed and the order of guessing.

The former are represented as Fil and Fil_N. The relations encoded for Fil are the following:

1. Model the propagation from plaintext and ciphertext,

$$\texttt{Inv\_St}[i] = \texttt{DirProp}(\texttt{St}[i-1]),\ \forall i \in \{1, \dots, n_I\}$$
$$\texttt{Inv\_St}[i] = \texttt{InvProp}(\texttt{St}[i+1]),\ \forall i \in \{n_F, \dots, n_r - 1\}$$

2. Identify the filtering steps bytes and their number of differences,

$$\texttt{Fil}[i,j] = \left|\left\{k \in \{0,1,\dots,7\} : \texttt{Inv\_St}[i,j,k] = 2\ \wedge\ \texttt{St}[i,j,k] \neq 2\right\}\right|$$

3. Deduce all the bytes involved in their evaluations,

$$\forall i,j \in \{0,\dots,n_r\} \times \{0,\dots,7\},\ \texttt{Fil}[i,j] \neq 0 \implies \texttt{Fil\_N}[i,j] = 1$$
$$\forall i,j \in \{1,\dots,n_I\} \times \{0,\dots,3\} :$$
$$\begin{cases} \texttt{Fil\_N}[i,2j+1] = 1 \implies \texttt{Fil\_N}[i-1,(2j+2) \mod 8] = 1 \\ \texttt{Fil\_N}[i,2j+1] = 1 \implies \texttt{Fil\_N}[i-1,(2j+3) \mod 8] = 1 \\ \texttt{Fil\_N}[i,2j] = 1 \implies \texttt{Fil\_N}[i-1,2j+1] = 1 \end{cases}$$
$$\forall i,j \in \{n_F,\dots,n_r-1\} \times \{0,\dots,3\} :$$
$$\begin{cases} \texttt{Fil\_N}[i,2j] = 1 \implies \texttt{Fil\_N}[i+1,(2j-1) \mod 8] = 1 \\ \texttt{Fil\_N}[i,2j] = 1 \implies \texttt{Fil\_N}[i+1,2j] = 1 \\ \texttt{Fil\_N}[i,2j+1] = 1 \implies \texttt{Fil\_N}[i+1,2j] = 1 \end{cases}$$

For the order of the guessing, we need the permutation of the key bytes, its inverse, and the maximum index for the evaluation of each guessed byte:

1. Model the permutation, using the MiniZinc global constraint `all_different`,

$$\texttt{all\_different}(KP)$$

2. Model the inverse of the permutation,

$$\forall i \in \{0,\dots,15\}\ \texttt{KP\_id}[KP[i]] = i+1$$

3. Initialize the max index according to the initial and final transformation,

$$\begin{cases} \texttt{MK}[0,0] = 0 \wedge \texttt{MK}[0,1] = \texttt{KP\_id}[15] \\ \texttt{MK}[0,2] = 0 \wedge \texttt{MK}[0,3] = \texttt{KP\_id}[15] \\ \texttt{MK}[0,4] = 0 \wedge \texttt{MK}[0,5] = \texttt{KP\_id}[15] \\ \texttt{MK}[0,6] = 0 \wedge \texttt{MK}[0,7] = \texttt{KP\_id}[15] \\ \texttt{MK}[n_r,0] = \texttt{KP\_id}[3] \wedge \texttt{MK}[n_r,1] = 0 \\ \texttt{MK}[n_r,2] = \texttt{KP\_id}[2] \wedge \texttt{MK}[n_r,3] = 0 \\ \texttt{MK}[n_r,4] = \texttt{KP\_id}[1] \wedge \texttt{MK}[n_r,5] = 0 \\ \texttt{MK}[n_r,6] = \texttt{KP\_id}[0] \wedge \texttt{MK}[n_r,7] = 0 \end{cases}$$

4. Model the max index,

$$\forall i,j \in \{1,\dots,n_I\} \times \{0,\dots,3\} :$$
$$\begin{cases} \texttt{MK}[i,2j] = \texttt{MK}[i-1,2j+1] \\ \texttt{MK}[i,2j+1] = \max(\texttt{MK}[i-1,2j+2], \texttt{MK}[i-1,2j+3], SK_i^j) \end{cases}$$
$$\forall i,j \in \{n_F,\dots,n_r-1\} \times \{0,\dots,3\} :$$
$$\begin{cases} \texttt{MK}[i,2j+1] = \texttt{MK}[i+1,2j] \\ \texttt{MK}[i,2j] = \max(\texttt{MK}[i+1,2j-1], \texttt{MK}[i+1,2j], SK_i^j) \end{cases}$$

Finally, we can compute the ($\log_2$ of the) number of pairs eliminated before each guessing step:

$$\forall k \in \{0,\dots,15\},\ P\_El[k] = \sum_{i=0}^{n_r}\sum_{j=0}^{7}(\texttt{Fil}[i,j] \cdot \mathbb{1}_{\texttt{Fil}[i,j]>0 \wedge \texttt{MK}[i,j]\leq k})$$

where $\mathbb{1}$ is the indicator function.

7.2 Complexity evaluation and Objective Function

The data complexity evaluation is similar to [HSE23]. In practice, we implement the complexity of the pairs elimination phase with the set $C_{guess} = (8(i+1) - P\_El[i])$. The heaviest complexity step is the maximum over i of $C_{guess}[i] + \log_2(|\{j \ s.t.\ C_{guess}[j] = C_{guess}[i]\}|)$, where for the logarithm we use the approximation $\log_2(x) = 2.2x - 2$ as in [BGG+23b].
For the final exhaustive search, the complexity is as shown in [ÖVTK09],

$$\log_2\left(2^{128}\cdot\left(1-\frac{2^f}{2^{8\cdot(\max(MK)-1)}}\right)^{2^{IP-P\_El[15]}}\right)$$

where f is the number of bits of the last filtering byte where a fixed known difference has to be forced and 2^{IP} is the number of initial pairs. We approximate the function as

$$\log_2\left(2^{128}\left(1-\frac{1}{2^k}\right)^{2^n}\right) \approx \log_2\left(2^{128}\cdot e^{-2^{n-k}}\right) = 128 - 2^{n-k}\log_2(e)$$

7.2.1 Modeling the Hash Tables

The main improvement of our automatic model compared to others in the literature is the automation of the *hash tables approach* and its integration with the model of the distinguisher search.

Let $IT_1, IT_2, \dots, IT_T$ denote T tables with binary elements, representing the modeling of T hash tables. A 1 means that the byte in that position will be guessed in the corresponding hash table. Another set of T tables, $OT_1, OT_2, \dots, OT_T$, represents the bytes that are computed during the creation of the hash table. Among the bytes leading to pairs eliminations, we select T of them as representatives of the hash tables. The constraints used for these tables (apart from the relations among the bytes) represent two properties. First, every selected filtering byte must be active in at least one of the second set of tables. Moreover, as conditions for the first set of tables, given the filtering bytes related to a hash table, we set as 1 the nearest bytes to the filtering tables, which are involved in its computation and have already been evaluated as outputs in at least one previous table or in the pairs elimination part. An important addition to the model is the modification of the objective function. Next, the pairs filtering must be modeled only on the filtering bytes not involved in the hash tables, thus drastically reducing this part of the total complexity. On the other hand we have to take into account the generation of the tables and the memory accesses. Regarding the $\log_2$ of the first part of the evaluation, we have that

$$\max_{i=1}^{T}(8\cdot(\sum_{k=0}^{n_r}\sum_{j=0}^{7}(IT_i[k,j]) + TK_i))$$

where TK_i is the number of key bytes to be guessed in the computation of table i. For the second part we count one access as a quarter round of encryption, as in [AAA+18]. Each hash table is represented by some filtering bytes, which involve a total of CT_i bit conditions. Moreover, each table also determines as output some key bytes, denoted KT_i.

Each table is then accessed a total number of times equal to the number of remaining pairs 2^{RP}, after filtering multiplied by 2 to the power of the sum of the numbers of the free bits after accessing the previous tables:

$$A_i = 2^{RP+\sum_{j=1}^{i-1}(8\cdot KT_i - CT_i)}$$

The $\log_2$ of the tables' access complexity is approximately

$$\max_{i=1}^{T}(RP + \sum_{j=1}^{i-1}(8 \cdot KT_i - CT_i)) \, .$$

7.3 Results: improved single-key recovery attack

Using our model, we improve the attack complexity on 27-rounds HIGHT in the single-key setting with initial and final transformations. The extension of the trail can be found in Table 6, and the complete trail in Appendix G. The hash tables filtering bytes are highlighted in red, and one of the pairs elimination processes is in blue. The bytes to be guessed in the pairs elimination process are highlighted in green. A detailed review of the attack procedure can be found in Appendix H, where the input bytes of each table have already been guessed or evaluated in previous steps, either the pairs elimination phase or in previous tables. We first detail the filtering process and give a description of the hash tables computation. The permutation of the key bytes guessed in the pairs elimination phase is $(0, 3, 13, 12)$, while the hash tables involve the following filtering bytes (in order of computation): $(S^2_{28}), (S^3_6), (S^4_{27}), (S^3_8), (S^3_7), (S^6_{26}), (S^0_{25}), (S^2_{25}, S^2_{24})$,

Table 6: Our impossible differential trail with extensions for a 27 rounds attack on HIGHT

PT	???????0	10000000	00000000	00000000	????????	????????	????????	????????	
R3	???????0	10000000	00000000	00000000	????????	????????	????????	????????	↑
R4	10000000	00000000	00000000	00000000	????????	????????	????????	???????1	↑
R5	00000000	00000000	00000000	00000000	????????	????????	???????1	10000000	↑
R6	00000000	00000000	00000000	00000000	????????	???????1	10000000	00000000	↑
R7	00000000	00000000	00000000	00000000	???????1	10000000	00000000	00000000	↑
R8	00000000	00000000	00000000	00000000	10000000	00000000	00000000	00000000	↑
				ID	**Trail**				
R24	00000000	00000000	00000000	??1????1	00000000	00000000	00000000	00000000	↓
R25	00000000	????????	??1????1	00000000	00000000	00000000	00000000	00000000	↓
R26	????????	???????1	00000000	00000000	00000000	00000000	00000000	????????	↓
R27	???????1	00000000	00000000	00000000	00000000	????????	????????	????????	↓
R28	00000000	00000000	00000000	????????	????????	????????	????????	???????1	↓
R29	00000000	????????	????????	????????	????????	????????	???????1	00000000	↓
R30	????????	????????	????????	????????	????????	???????1	00000000	????????	↓
CT	????????	????????	????????	????????	????????	????????	???????1	00000000	

7.3.1 Complexity Evaluation

We have to generate 2^{IP} initial pairs satisfying the plaintext and ciphertext truncated differences. From a fixed 64-bit string we can construct 2^{39} plaintexts having the same bit in the positions where the plaintext difference is fixed and another 2^{39} such that the difference with the first ones satisfies the plaintext difference. These sets can generate a total of 2^{78} pairs. Each one of these pairs satisfies the ciphertext condition with probability 2^{-9}, so for each structure, a total of 2^{69} good pairs will remain. Therefore we will need 2^{IP-69} structures of $2^{39} + 2^{39} = 2^{40}$ plaintexts each, resulting in a total amount of 2^{IP-29} initial plaintexts for our attack. A summary of the complexity evaluation of the attack follows (we count C_B as a quarter round encryption, $C_B = \frac{1}{108}C_E$):

- Initial pairs generation: the total complexity of the step is $C_0 = 2^{IP-29}C_E$
- Pairs elimination: the total complexity is the sum of the complexity of each guessing step, in our case $C_1 = 2 \cdot 2^{IP+16} \cdot 7C_B$
- Hash tables computation: the total complexity is the sum of the complexity of each step, whose leading term in our case is $C_2 \leq (2^8)^{14}C_B$
- Hash tables access: the total complexity is the sum of each the complexity of table access, in our case $C_3 = 2^{IP-24} \cdot 2^{32} \cdot (2^8 + 2^{16} + 2^{24} + 2^{24} + 2^{24} + 2^{32} + 2^{32} + 2^{32} + 2^{31})C_B$
- Exhaustive search: $C_4 = 2^{128} \cdot (1 - \frac{2^{31}}{2^{88}})^{2^{IP-24}}$

By setting $IP = 84$, the data complexity is 2^{55} plaintexts and the total time complexity is $C_T = C_0 + C_1 + C_2 + C_3 + C_4 \leq 2^{55}C_E + (2^{101} \cdot 7 + 2^{112} + 2^{125.33})C_B + 2^{116.5}C_E \leq 2^{118.9}C_E$.

8 Conclusions and future work

We presented an extension of previous techniques to find impossible differential trails by introducing a hybrid model. The model is available in multiple formalisms and combines the advantages of the cell-wise propagation analysis with the precision of the bit-wise approach, while also supporting probabilistic transitions in the related-key setting. We also introduced an effective technique to automatically invert symmetric ciphers. We demonstrated the generality of the techniques above by implementing them in an automated tool, namely CLAASP. We effectively applied the tool to the LBlock cipher to not only recover state-of-the-art results for 16 rounds but also expand them with new 16-round impossible differentials and a new 18-round improbable differential.

Additionally, we developed a new CP model to evaluate the complexity of ID attacks and automatically identify the optimal impossible differential trail for key recovery in the HIGHT block cipher. Our model improves the key-recovery attacks on 27-round HIGHT by identifying a new trail and leveraging the hash tables approach presented in [CWP12].

Future work will aim to generalize our ID attack model for HIGHT to enhance automated tools. This will involve expanding support for complexity evaluation of ID attacks for SAT, SMT, MILP, and CP, with the option to incorporate the hash tables approach. Another promising direction is integrating the differential clustering effect into the model for optimal distinguisher search, which is currently performed as a post-processing step in the hybrid probabilistic model.

References

[AAA+18] Seyyed Arash Azimi, Siavash Ahmadi, Zahra Ahmadian, Javad Mohajeri, and Mohammad Reza Aref. Improved impossible differential and biclique cryptanalysis of HIGHT. *Int. J. Commun. Syst.*, 31(1), 2018.

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The qarmav2 family of tweakable block ciphers. *IACR Trans. Symmetric Cryptol.*, 2023(3):25–73, 2023.

[ALP+19] Elena Andreeva, Virginie Lallemand, Antoon Purnal, Reza Reyhanitabar, Arnab Roy, and Damian Vizár. Forkcipher: A new primitive for authenticated encryption of very short messages. In *ASIACRYPT (2)*, volume 11922 of *Lecture Notes in Computer Science*, pages 153–182. Springer, 2019.

[AST$^+$17] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2017(4):99–129, 2017.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[BDD$^+$24] Christina Boura, Nicolas David, Patrick Derbez, Rachelle Heim Boissier, and María Naya-Plasencia. A generic algorithm for efficient key recovery in differential attacks - and its associated tool. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part I*, volume 14651 of *Lecture Notes in Computer Science*, pages 217–248. Springer, 2024.

[BFG$^+$24] Emanuele Bellini, Mattia Formenti, David Gérault, Juan Grados, Anna Hambitzer, Yun Ju Huang, Paul Huynh, Mohamed Rachidi, Raghvendra Rohit, and Sharwan K. Tiwari. Claasping ARADI: automated analysis of the ARADI block cipher. *IACR Cryptol. ePrint Arch.*, page 1324, 2024.

[BGG$^+$23a] Emanuele Bellini, David Gérault, Juan Grados, Yun Ju Huang, Rusydi H. Makarim, Mohamed Rachidi, and Sharwan K. Tiwari. CLAASP: A cryptographic library for the automated analysis of symmetric primitives. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers*, volume 14201 of *Lecture Notes in Computer Science*, pages 387–408. Springer, 2023.

[BGG$^+$23b] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin. Fully automated differential-linear attacks against ARX ciphers. In Mike Rosulek, editor, *Topics in Cryptology - CT-RSA 2023 - Cryptographers' Track at the RSA Conference 2023, San Francisco, CA, USA, April 24-27, 2023, Proceedings*, volume 13871 of *Lecture Notes in Computer Science*, pages 252–276. Springer, 2023.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BMNS14] Christina Boura, Marine Minier, María Naya-Plasencia, and Valentin Suder. Improved impossible differential attacks against round-reduced lblock. *IACR Cryptol. ePrint Arch.*, page 279, 2014.

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: Applications to clefia, camellia, lblock and simon. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 179–199. Springer, 2014.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *J. Cryptology*, 4:3–72, 1991.

[CJF$^+$16] Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. New automatic search tool for impossible differentials and zero-correlation linear approximations. *IACR Cryptol. ePrint Arch.*, page 689, 2016.

[CWP12] Jiazhe Chen, Meiqin Wang, and Bart Preneel. Impossible differential cryptanalysis of the lightweight block ciphers tea, XTEA and HIGHT. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 117–137. Springer, 2012.

[CZZ22] Weiwei Cao, Wentao Zhang, and Chunning Zhou. New automatic search tool for searching for impossible differentials using undisturbed bits. In Yi Deng and Moti Yung, editors, *Information Security and Cryptology - 18th International Conference, Inscrypt 2022, Beijing, China, December 11-13, 2022, Revised Selected Papers*, volume 13837 of *Lecture Notes in Computer Science*, pages 43–63. Springer, 2022.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[GMS16] David Gérault, Marine Minier, and Christine Solnon. Constraint programming models for chosen key differential cryptanalysis. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 584–601. Springer, 2016.

[HGSE24] Hosein Hadipour, Simon Gerhalter, Sadegh Sadeghi, and Maria Eichlseder. Improved search for integral, impossible differential and zero-correlation attacks application to ascon, forkskinny, skinny, mantis, PRESENT and qarmav2. *IACR Trans. Symmetric Cryptol.*, 2024(1):234–325, 2024.

[HSE23] Hosein Hadipour, Sadegh Sadeghi, and Maria Eichlseder. Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*, volume 14007 of *Lecture Notes in Computer Science*, pages 128–157. Springer, 2023.

[HSH$^+$06] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International*

Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[KDH12] Ferhat Karakoç, Hüseyin Demirci, and A. Emre Harmanci. Impossible differential cryptanalysis of reduced-round lblock. In Ioannis G. Askoxylakis, Henrich Christopher Pöhls, and Joachim Posegga, editors, *Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems - 6th IFIP WG 11.2 International Workshop, WISTP 2012, Egham, UK, June 20-22, 2012. Proceedings*, volume 7322 of *Lecture Notes in Computer Science*, pages 179–188. Springer, 2012.

[KHS$^+$03] Jongsung Kim, Seokhie Hong, Jaechul Sung, Changhoon Lee, and Sangjin Lee. Impossible differential cryptanalysis for block cipher structures. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, volume 2904 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 2003.

[Knu98] Lars Knudsen. Deal-a 128-bit block cipher. *Complexity*, 258(2), 1998.

[LDKK08] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. New impossible differential attacks on AES. In *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008.

[Lu07] Jiqiang Lu. Cryptanalysis of reduced versions of the HIGHT block cipher from CHES 2006. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings*, volume 4817 of *Lecture Notes in Computer Science*, pages 11–26. Springer, 2007.

[MN12] Marine Minier and María Naya-Plasencia. A related key impossible differential attack against 22 rounds of the lightweight block cipher lblock. *Inf. Process. Lett.*, 112(16):624–629, 2012.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[NSB$^+$07] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In Christian Bessiere, editor, *CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[ÖVTK09] Onur Özen, Kerem Varici, Cihangir Tezcan, and Çelebi Kocair. Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT. In Colin Boyd and Juan Manuel González Nieto, editors, *Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Brisbane, Australia, July 1-3, 2009, Proceedings*, volume 5594 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.

[RCT13] Saeed Rostami, Sadegh Bamohabbat Chafjiri, and Seyed Amir Hossein Tabatabaei. Related-key impossible differential cryptanalysis of full-round HIGHT. In Pierangela Samarati, editor, *SECRYPT 2013 - Proceedings of* *the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 537–542. SciTePress, 2013.

[SGWW20] Ling Sun, David Gérault, Wei Wang, and Meiqin Wang. On the usage of deterministic (related-key) truncated differentials and multidimensional linear approximations for SPN ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(3):262–287, 2020.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 185–215, 2017.

[Tez14] Cihangir Tezcan. Improbable differential attacks on present using undisturbed bits. *J. Comput. Appl. Math.*, 259:503–511, 2014.

[WWZ14] Long Wen, Meiqin Wang, and Jingyuan Zhao. Related-key impossible differential attack on reduced-round lblock. *J. Comput. Sci. Technol.*, 29(1):165–176, 2014.

[WZ11] Wenling Wu and Lei Zhang. Lblock: A lightweight block cipher. In Javier López and Gene Tsudik, editors, *Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings*, volume 6715 of *Lecture Notes in Computer Science*, pages 327–344, 2011.

A Description of LBlock

LBlock is a lightweight block cipher proposed by Wu and Zhang at ACNS 2011 [WZ11]. The round function acts on a 64-bit state. It follows a two-branched Feistel structure, where the receiving branch of the Feistel function is first rotated by 8 bits. The inner round function is made of one round key addition, a parallel application of 8 different 4-bit S-boxes and a nibble-wise permutation.

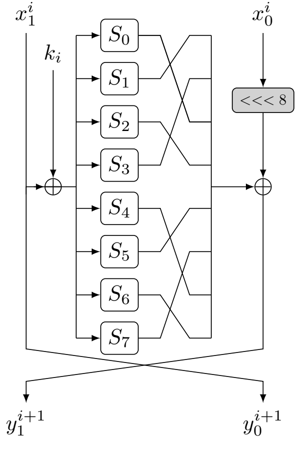

Figure 2: One round of the LBlock cipher.

The 80-bit master key K is stored in a key register and denoted as $K = k_{79}k_{78}\cdots k_1k_0$. At each round i, the 32-bit round key sk_i consists of the 64 leftmost bits of the current key state: $k_i = k_{79\sim 48}$. The key update is described in algorithm 2.

Algorithm 2: Key schedule of LBlock

```
$sk_0 = K_{79\sim 48}$;
for $1 \leq r \leq 31$ do
    $k_{79\sim 0} \leftarrow k_{79\sim 0} \lll 29$;
    $k_{79\sim 76} \leftarrow S_9(k_{79\sim 76})$;
    $k_{75\sim 72} \leftarrow S_8(k_{75\sim 72})$;
    $k_{50\sim 47} \leftarrow k_{50\sim 47} \oplus [i]_2$;
    $sk_{5r} \leftarrow k_{79\sim 48}$;
end
```

B Description of HIGHT

HIGHT is a lightweight block cipher proposed by Hong et al. at CHES 2006 [HSH+06]. The round function acts on a 64-bit state. Its round function is some sort of a concatenation of 4 Feistel structures acting on pairs of bytes which are then rotated. The key is inducted alternately on the bytes by XOR and modular addition, while the state operations are rotations and either XOR, if the key is used with the modular addition, or modular addition. The 128-bit master key K is used for an initial transformation and then at each round the 32-bit subkey involved in the state is evaluated via modular addition with fixed constants of previous subkeys.
The 128-bit initial key MK is used for the computation of the 32-bit round keys SK_i and the initial and final whitening keys WK as shown in algorithm 3 and algorithm 4

Algorithm 3: Whitening key schedule of HIGHT

```
for $0 \leq i \leq 3$ do
    $WK_i \leftarrow MK_{i+12}$;
end
for $4 \leq i \leq 7$ do
    $WK_i \leftarrow MK_{i-4}$;
end
```

The initial and final whitening are performed at the beginning and end of the encryption by adding or XORing the whitening key to the plaintext and to the last round output respectively, as shown in algorithm 5. From now on we will indicate the i-th byte of the r-th round output as S_r^i.

The ENC(S) encryption function is evaluated by the application of a round function 32 times. The round function is described in algorithm 6. This function needs the definition of two auxiliary functions, F_0 and F_1, which are defined as follows:

$$F_0(x) = (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 7)$$

$$F_1(x) = (x \lll 3) \oplus (x \lll 4) \oplus (x \lll 6)$$

A graphic representation of the round function can be found in Figure 3.

Algorithm 4: Key schedule of HIGHT

```
$\delta_0 \leftarrow 1011010$;
for $1 \leq i \leq 127$ do
    $\delta_i \leftarrow ((\delta_{i-1} \oplus (\delta_{i-1} \gg 3)) \ll 6) \oplus (\delta_{i-1} \gg 1)$;
end
for $0 \leq i \leq 7$ do
    for $0 \leq j \leq 7$ do
        $SK_{16i+j} \leftarrow MK_{j-1 \mod 8} \boxplus \delta_{16i+j}$;
    end
    for $0 \leq j \leq 7$ do
        $SK_{16i+j+8} \leftarrow MK_{(j-1 \mod 8)+8} \boxplus \delta_{16i+j+8}$;
    end
end
```

Algorithm 5: Initial and final whitening of HIGHT

```
$S_0^0 \leftarrow P_0$; $S_0^1 \leftarrow P_1 \oplus WK_3$; $S_0^2 \leftarrow P_2$; $S_0^3 \leftarrow P_3 \boxplus WK_2$; $S_0^4 \leftarrow P_4$;
  $S_0^5 \leftarrow P_5 \oplus WK_1$; $S_0^6 \leftarrow P_6$; $S_0^7 \leftarrow P_7 \boxplus WK_0$;
ENC(S);
$C_0 \leftarrow S_{32}^7$; $C_1 \leftarrow S_{32}^0 \oplus WK_7$; $C_2 \leftarrow S_{32}^1$; $C_3 \leftarrow S_{32}^2 \boxplus WK_6$; $C_4 \leftarrow S_{32}^3$;
  $C_5 \leftarrow S_{32}^4 \oplus WK_5$; $C_6 \leftarrow S_{32}^5$; $C_7 \leftarrow S_{32}^6 \boxplus WK_4$;
```

Algorithm 6: Round function of HIGHT

```
for $0 \leq r \leq 31$ do
    for $0 \leq i \leq 3$ do
        $S_{r+1}^{2i} \leftarrow S_r^{2i+1}$
    end
    for $0 \leq i \leq 1$ do
        $S_{r+1}^{4i-1 \mod 8} \leftarrow S_r^{4i} \oplus (F_1(S_{r-1}^{4i+1}) \boxplus SK_{4r+3-2i})$
    end
    for $0 \leq i \leq 1$ do
        $S_{r+1}^{4i+1} \leftarrow S_r^{4i+2} \boxplus (F_1(S_{r-1}^{4i+3}) \oplus SK_{4r+2-2i})$
    end
end
```

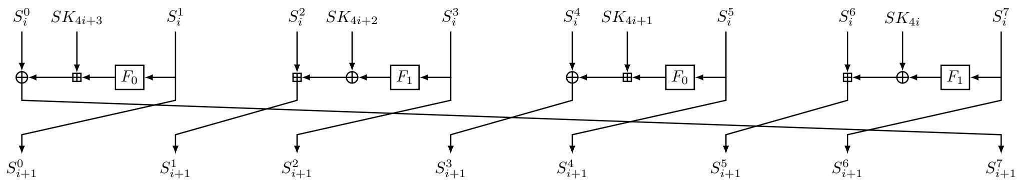

Figure 3: One round of the HIGHT cipher.

C CLAASP automatic cipher inversion: a toy example

```
cipher = {
...
'cipher_rounds' : [
# round 0
[{ # round = 0 - round component = 0
   'id': 'sbox_0_0',
   'type': 'sbox',
   'input_bit_size': 3,
   'input_id_link': ['plaintext'],
   'input_bit_positions': [[0, 1, 2]],
   'output_bit_size': 3,
    'description': [0, 5, 3, 2, 6, 1, 4, 7]},
 { # round = 0 - round component = 1
   'id': 'xor_0_1',
   'type': 'word_operation',
   'input_bit_size': 6,
   'input_id_link': ['sbox_0_0', 'plaintext'],
   'input_bit_positions': [[0, 1, 2], [3, 4, 5]],
   'output_bit_size': 3,
   'description': ['XOR', 2]},
 { # round = 0 - round component = 2
   'id': 'cipher_output_0_2',
   'type': 'cipher_output',
   'input_bit_size': 6,
   'input_id_link': ['xor_0_1', 'plaintext'],
   'input_bit_positions': [[0, 1, 2], [0, 1, 2]],
   'output_bit_size': 6,
   'description': ['cipher_output'],
 }]]
}
```

Listing 1: CLAASP representation of the toy example

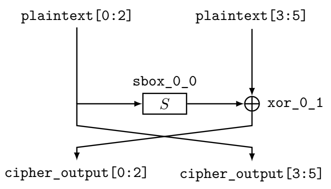

Figure 4: Toy example to illustrate the cipher inversion.

Consider the example depicted in Figure 4 and its CLAASP object (Listing 1):

Step 1. $L =$ [`cipher_output_0_2`]. The bits `plaintext[0:2]` can be obtained as they are equal to `cipher_output_0_2[3:5]`.

Step 2. $L =$ [`cipher_output_0_2`, `plaintext[0:2]`]. The S-box `sbox_0_0` can be **evaluated** as its input bits `plaintext[0:2]` are available.

Step 3. $L =$ [`cipher_output_0_2`, `plaintext[0:2]`, `sbox_0_0`]. The *XOR* operation `xor_0_1` can be **inverted** as its output `cipher_output_0_2[0:2]` and its second input `sbox_0_0` are available.

Step 4. $L =$ [`cipher_output_0_2`, `plaintext[0:2]`, `sbox_0_0`, `inv_xor_0_1`]. Finally, the last bits `plaintext[3:5]` can be obtained as they are equal to the output of the inverted XOR.

D 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ found for LBlock by [CJF+16]

Table 7: Deterministic subkeys used in the 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ of [CJF+16].

0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0010	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
?1??	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	00?1	??00	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	000?	1??0	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	?1??	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	00?1

Table 8: State of the the 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ of [CJF+16] using a bit-wise model.

0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	???1	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	????	0000	0000	0000	0000	0000	0000	0000	0000	0000	???1	0000	0000	0000	0000
????	???1	0000	0000	0000	0000	0000	0000	0000	????	0000	0000	0000	0000	0000	0000
????	0000	????	0000	0000	0000	0000	????	????	???1	0000	0000	0000	0000	0000	0000
0000	0000	????	????	0000	????	????	???1	????	0000	????	0000	0000	0000	0000	????
????	????	0000	????	????	????	????	????	0000	0000	????	????	0000	????	????	???1
????	????	????	????	????	????	????	????	????	????	????	????	????	????	????	????
????	????	????	????	????	????	????	????	????	????	????	????	????	????	????	????
0000	????	????	????	0000	????	0000	????	????	????	????	????	????	????	????	????
0000	????	????	0000	0000	0000	????	????	0000	????	????	????	0000	????	0000	????
0000	????	0000	0000	0000	????	0000	????	0000	????	????	0000	0000	0000	????	????
0000	0000	0000	0000	0000	0000	0000	????	0000	????	0000	0000	0000	????	0000	????
0000	0000	0000	0000	0000	0000	0000	????	0000	0000	0000	0000	0000	0000	0000	????
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	????
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Table 10: Evolution of the LBlock key schedule over 16 rounds, when $\Delta K = 0x40$, in the bit-wise deterministic truncated model. The differential $(000000000, 00000000) \xrightarrow[\Delta K = 0x40]{16r} (00000000, 00000000)$ is impossible due to a contradiction at byte 11 of round 9 if $\alpha = 0$ and at byte 15 of round 9 otherwise.

Round	Subkey							
0	0000	0000	0000	0000	0000	0000	0000	0000
1	0000	0000	0000	0000	0000	0000	0000	0000
2	0000	0000	0000	0001	0000	0000	0000	0000
3	0000	0000	0000	0000	0000	0000	0000	0000
4	0000	0000	0000	0000	0000	0000	0000	0000
5	0000	0000	1000	0000	0000	0000	0000	0000
6	0000	0000	0000	0000	0000	0000	0000	0000
7	0000	0000	0000	0000	0000	0000	0000	0010
8	$\alpha\beta\gamma\delta$	0000	0000	0000	0000	0000	0000	0000
9	0000	0000	0000	0000	0000	0000	0000	0000
10	0000	0000	0000	0000	0000	$00\alpha\beta$	$\gamma\delta00$	0000
11	0000	0000	0000	0000	0000	0000	0000	0000
12	0000	0000	0000	0000	0000	0000	0000	0000
13	0000	0000	0000	000α	$\beta\gamma\delta0$	0000	0000	0000
14	0000	0000	0000	0000	0000	0000	0000	0000
15	0000	0000	0000	0000	0000	0000	0000	0000

Table 9: State of the 16-round impossible differential $0 \xrightarrow{\Delta k_{11}=1} 0$ of [CJF+16] using the hybrid model.

0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	2222	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	5555	0000	0000	0000	0000	0000	0000	0000	0000	0000	2222	0000	0000	0000	0000
4444	2222	0000	0000	0000	0000	0000	0000	0000	5555	0000	0000	0000	0000	0000	0000
6666	0000	????	0000	0000	0000	0000	5555	4444	2222	0000	0000	0000	0000	0000	0000
0000	0000	0000	????	0000	2222	4444	2222	6666	0000	????	0000	0000	0000	0000	5555
????	????	0000	5555	????	????	6666	????	0000	0000	0000	????	0000	2222	4444	2222
4444	????	0000	????	????	????	5555	????	????	????	????	????	????	????	????	????
????	0000	????	????	6666	4444	0000	????	4444	????	0000	????	????	????	5555	????
0000	????	4444	6666	0000	6666	0000	????	????	0000	????	????	6666	4444	0000	????
0000	4444	6666	0000	0000	0000	????	0000	0000	????	4444	6666	0000	6666	0000	????
0000	6666	0000	0000	0000	????	0000	2222	0000	4444	6666	0000	0000	0000	????	0000
0000	0000	0000	0000	0000	0000	0000	4444	0000	6666	0000	0000	0000	????	0000	2222
0000	0000	0000	0000	0000	0000	0000	6666	0000	0000	0000	0000	0000	0000	0000	4444
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	6666
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Listing 2: CLAASP script to find 16-round impossible trails with the hybrid model, in the related-key setting

```
from claasp.cipher_modules.models.cp.mzn_models.
    mzn_hybrid_impossible_xor_differential_model import
    MznHybridImpossibleXorDifferentialModel
from claasp.ciphers.block_ciphers.lblock_block_cipher import LBlockBlockCipher
from claasp.cipher_modules.models.utils import set_fixed_variables, integer_to_bit_list
lblock = LBlockBlockCipher(number_of_rounds=16)
mzn = MznHybridImpossibleXorDifferentialModel(lblock)
fixed_variables = [set_fixed_variables('key', 'not_equal', range(80), [0]*80)]
trails = mzn.find_all_impossible_xor_differential_trails(16, fixed_variables, 'Chuffed'
    , 1, 8, 16, intermediate_components=False)
```

E 18-round improbable differential for LBlock

Table 11 shows the 18-round improbable trail, and the patterns for the subkeys are given in Table 12.

Table 11: Our 18-round improbable trail. We use a truncated cell-wise notation: 0 indicates a null word difference, * represents an unknown but nonzero difference, and ? denotes a fully undetermined difference.

State (truncated)	Subkey
(00000000, 00000008)	00000000
(00000800, 00000000)	00000800
(00000000, 00000800)	00000000
(00080000, 00000000)	00000000
(0*000000, 00080000)	00040000
(*?000000, 0*000000)	00000000
(?0*0000*, *?000000)	00000000
(00?*0**?, ?0*0000*)	0*000000
(?*0?*??*, 00?*0**?)	00000000
(???*??*?, ?*0?*??*)	000000?*
(?*?*?**?, ??*?????)	
(0?0*?***, ?*?*?**?)	α0000000
(?00*0*?0, 0?0*?***)	00000000
(000*0*00, ?00*0*?0)	0000?*00
(000*0000, 000*0*00)	00000000
(000*0000, 000*0000)	00000000
(00000000, 000*0000)	000*0000
(00000000, 00000000)	00000000
(00000000, 00000000)	00000000
(00000000, 00000000)	

Table 12: The conditional subkeys used for the 18-round improbable differential. We use a truncated cell-wise notation: 0 indicates a null word difference, * represents an unknown but nonzero difference, and ? denotes a fully undetermined difference. Here α equals either 0x0 or 0x8.

Round	Deterministic Subkeys	Conditional Subkeys
0	00000000	00000000
1	00000800	00000800
2	00000000	00000000
3	00000000	00000000
4	00040000	00040000
5	00000000	00000000
6	00000000	00000000
7	0*000000	0*000000
8	00000000	00000000
9	000000?*	000000?*
10	?0000000	α0000000
11	00000000	00000000
12	0000?*?0	0000?*00
13	00000000	00000000
14	00000000	00000000
15	000*?000	000*0000
16	00000000	00000000
17	00000000	00000000

Table 13 shows the state of the 18-round improbable trail for LBlock under the bit-wise model. The backward propagation leading to byte X_{10} of round 9 is highlighted in color. The shade changes whenever the difference goes through an S-box. Information loss can be seen between the transition from ???1 $\xrightarrow{S_4}$???? when computing X_{11} of round 14.

Table 13: State of the 18-round improbable differential described in Table 11 using a bit-wise model.

State																Subkey
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	1000	00000000
0000	0000	0000	0000	0000	1000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	00000800
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	1000	0000	0000	00000000
0000	0000	0000	1000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	00000000
0000	1???	0000	0000	0000	0000	0000	0000	0000	0000	0000	1000	0000	0000	0000	0000	00040000
????	????	0000	0000	0000	0000	0000	0000	0000	1???	0000	0000	0000	0000	0000	0000	00000000
????	0000	????	0000	0000	0000	0000	1???	????	????	0000	0000	0000	0000	0000	0000	00000000
0000	0000	????	????	0000	????	????	????	????	0000	????	0000	0000	0000	0000	1???	02000000
????	????	0000	????	????	????	????	????	0000	0000	????	????	0000	????	????	????	00000000
????	????	????	????	????	????	????	????	????	????	**0000**	????	????	????	????	????	00000008
????	????	????	????	0000	????	????	????	????	????	**????**	????	????	????	????	????	
0000	????	0000	????	0000	????	????	???1	????	????	????	????	0000	????	????	????	00000000
0000	0000	0000	????	0000	????	????	0000	0000	????	0000	????	0000	????	????	???1	00000000
0000	0000	0000	????	0000	???1	0000	0000	0000	0000	0000	????	0000	????	????	0000	00000400
0000	0000	0000	????	0000	0000	0000	0000	0000	0000	0000	????	0000	???1	0000	0000	00000000
0000	0000	0000	???1	0000	0000	0000	0000	0000	0000	0000	????	0000	0000	0000	0000	00000000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	???1	0000	0000	0000	0000	00020000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	00000000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	00000000
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	

Listing 3: CLAASP script to verify the improbable differential

```
from claasp.ciphers.block_ciphers.lblock_block_cipher import LBlockBlockCipher
lblock = LBlockBlockCipher(number_of_rounds=18)
from claasp.cipher_modules.models.utils import set_fixed_variables, integer_to_bit_list
from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model import
    SatXorDifferentialModel
sat = SatXorDifferentialModel(lblock)
key = set_fixed_variables(component_id='key', constraint_type='equal', bit_positions=
    range(80), bit_values=[0] * 49 + [1] + [0]*30)
pt = set_fixed_variables(component_id='plaintext', constraint_type='equal',
    bit_positions= range(64), bit_values= [0] * 60 + [1,0,0,0])
ct = set_fixed_variables(component_id='intermediate_output_17_12', constraint_type='
    equal', bit_positions= range(64), bit_values= [0] * 64)
key_proba = [set_fixed_variables(component_id='intermediate_output_10_0',
    constraint_type='equal', bit_positions = range(1,4), bit_values= [0]*3)]
trail = sat.find_one_xor_differential_trail(fixed_values=key_proba + [key, pt, ct])
```

Listing 4: CLAASP script to verify the probability of the 18-round differential

```
from claasp.cipher_modules.models.cp.mzn_models.
    mzn_hybrid_impossible_xor_differential_model import
    MznHybridImpossibleXorDifferentialModel
from claasp.ciphers.block_ciphers.lblock_block_cipher import LBlockBlockCipher
from claasp.cipher_modules.models.utils import set_fixed_variables
lblock = LBlockBlockCipher(number_of_rounds=18)
model = MznHybridImpossibleXorDifferentialModel(lblock)
fixed_variables = [set_fixed_variables(component_id='key', constraint_type='equal',
    bit_positions=range(80), [0]*49+[1]+[0]*30)]
fixed_variables.append(set_fixed_variables(component_id='plaintext', constraint_type='
    equal', range(64), [0]*60 +[1,0,0,0]))
fixed_variables.append(set_fixed_variables('inverse_cipher_output_17_19', 'equal',
    range(64), [0]*64))
trails = model.find_all_impossible_xor_differential_trails(18, fixed_variables, '
    Chuffed', 1, 9, 18, intermediate_components=False, probabilistic=True)
```

F MiniZinc predicates for the HIGHT model

```
function array[int] of var 0..2: Xor2(array[int] of var 0..2: a, array[int] of var
    0..2: b)=
array1d(0..(length(a)-1), [if (a[j] == 2 \/ b[j] == 2) then 2 else ((a[j]+b[j]) mod 2)
    endif | j in 0..(length(a)-1)]);
```

Listing 5: The MiniZinc predicate representing the XOR of two bits

```
predicate Modadd_2(array[int] of var 0..2: a, array[int] of var 0..2: b,array[int] of
    var 0..2: c) = (
  let {
    array [0..length(a)-1] of var 0..2: as = LShift(a,1),
    array [0..length(a)-1] of var 0..2: bs = LShift(b,1),
    array [0..length(a)-1] of var 0..2: cs = LShift(c,1),
    var 0..length(a)-1: pivot;
  } in
  forall (i in 0..length(a)-1) (
    if i<pivot then c[i]=2 else (as[i]=0) /\ (bs[i]=0) /\ (cs[i]=0) endif
  ) /\
  xor_bit_p1_2(a[pivot],b[pivot],c[pivot])
  /\ if pivot>0 then a[pivot]+b[pivot]>0 else true endif
);
```

Listing 6: The MiniZinc predicate representing the Modular Addition of two words

```
function array[int] of var 0..2: LRot(array[int] of var 0..2: X, var int: val)=
array1d(0..(length(X)-1), [X[(j+val) mod length(X)] | j in 0..(length(X)-1)]);
```

Listing 7: The MiniZinc predicate representing the Left Rotation

G Our full 27 rounds HIGHT attack setting

Table 14: Our impossible differential trail with extensions for a 27 rounds attack on HIGHT

PT	???????0	10000000	00000000	00000000	????????	????????	????????	????????	
R3	???????0	10000000	00000000	00000000	????????	????????	????????	????????	↑
R4	10000000	00000000	00000000	00000000	????????	????????	????????	???????1	↑
R5	00000000	00000000	00000000	00000000	????????	????????	???????1	10000000	↑
R6	00000000	00000000	00000000	00000000	????????	???????1	10000000	00000000	↑
R7	00000000	00000000	00000000	00000000	???????1	10000000	00000000	00000000	↑
R8	00000000	00000000	00000000	00000000	10000000	00000000	00000000	00000000	↑
				Trail	**start**				
R9	00000000	00000000	00000000	10000000	00000000	00000000	00000000	00000000	↓
R10	00000000	?????100	10000000	00000000	00000000	00000000	00000000	00000000	↓
R11	?????100	10000000	00000000	00000000	00000000	00000000	00000000	????????	↓
R12	10000000	00000000	00000000	00000000	00000000	????????	????????	???????1	↓
R13	00000000	00000000	00000000	????????	????????	????????	???????1	10000000	↓
R14	00000000	????????	????????	????????	????????	???????1	10000000	00000000	↓
R15	????????	????????	????????	????????	???????1	10000000	00000000	????????	↓
R16	????????	????????	????????	???????0	10000000	????????	????????	????????	↓
				Middle	**point**				
R16	????????	????????	????????	???????1	????????	????????	????????	????????	↑
R17	????????	????????	???????1	00000000	????????	????????	????????	????????	↑
R18	????????	???????1	00000000	00000000	????????	????????	????????	????????	↑
R19	???????1	00000000	00000000	00000000	????????	????????	????????	????????	↑
R20	00000000	00000000	00000000	00000000	????????	????????	????????	???????1	↑
R21	00000000	00000000	00000000	00000000	????????	????????	???????1	00000000	↑
R22	00000000	00000000	00000000	00000000	????????	??1????1	00000000	00000000	↑
R23	00000000	00000000	00000000	00000000	??1????1	00000000	00000000	00000000	↑
				Trail	**end**				
R24	00000000	00000000	00000000	??1????1	00000000	00000000	00000000	00000000	↓
R25	00000000	????????	??1????1	00000000	00000000	00000000	00000000	00000000	↓
R26	????????	???????1	00000000	00000000	00000000	00000000	00000000	????????	↓
R27	???????1	00000000	00000000	00000000	00000000	????????	????????	????????	↓
R28	00000000	00000000	00000000	????????	????????	????????	????????	???????1	↓
R29	00000000	????????	????????	????????	????????	????????	???????1	00000000	↓
R30	????????	????????	????????	????????	????????	???????1	00000000	????????	↓
CT	????????	????????	????????	????????	????????	????????	???????1	00000000	

H Full procedure of the attack on 27-round HIGHT

Table 15: Impossible differential attack procedure on 27 rounds of HIGHT

Step	Description	Time complexity
	Pairs elimination	
1	Guess MK_0, MK_3 and compute S^0_{30}, S^0_{29} by partial decryption for each initial pair, then eliminate those whose difference does not propagate to the expected one (on average 1 pair every 2^8 will remain)	$2 \cdot 2^{IP} \cdot 2 \cdot 2^{16} C_B$
2	Guess MK_13 and compute S^5_3, S^3_4 by partial encryption for each remaining pair, filtering as before	$2 \cdot 2^{IP-8} \cdot 2 \cdot 2^{24} C_B$
3	Guess MK_12 and compute S^7_3, S^5_4, S^3_5 by partial encryption for each remaining pair, filtering as before	$2 \cdot 2^{IP-16} \cdot 3 \cdot 2^{32} C_B$
	Memory tables generation	
4	Build table T_0 by partial decryption, guessing bytes $C_1, \Delta C_1, C_2, \Delta C_2,$ $S^1_{29}, \Delta S^1_{29}, MK_2, MK_3, MK_7$. The condition $\Delta S^2_{28} = 0$ is a 8-bit condition, therefore if we index the table by $C_1, \Delta C_1, C_2, \Delta C_2, S^1_{29}, \Delta S^1_{29}, MK_3$ and take as outputs MK_2, MK_7, for each set of inputs we get on average 2^8 valid outputs.	$2 \cdot (2^8)^9 C_B$
5	Build table T_1 by partial encryption, guessing bytes $P_0, \Delta P_0, P_1, S^6_4, \Delta S^6_4,$ $S^4_5, \Delta S^4_5, MK_4, MK_7, MK_{15}$. The condition $\Delta S^3_6 = 0$ is a 8-bit condition, therefore if we index the table by $P_0, \Delta P_0, P_1, S^6_4, \Delta S^6_4, S^4_5, \Delta S^4_5, MK_7$ and take as outputs MK_4, MK_{15}, for each set of inputs we get on average 2^8 valid outputs.	$2 \cdot (2^8)^{10} C_B$
6	Build table T_2 by partial decryption, guessing bytes $C_3, \Delta C_3, C_4, \Delta C_4,$ $S^3_{29}, \Delta S^3_{29}, S^3_{28}, \Delta S^3_{28}, MK_1, MK_2, MK_6, MK_{15}$. The condition $\Delta S^4_{27} = 0$ is a 8-bit condition, therefore if we index the table by $C_3, \Delta C_3, C_4, \Delta C_4, S^3_{29}, \Delta S^3_{29}, S^3_{28}, \Delta S^3_{28}, MK_2, MK_{15}$ and take as outputs MK_1, MK_6, for each set of inputs we get on average 2^8 valid outputs.	$2 \cdot (2^8)^{12} C_B$
7	Build table T_3 by partial encryption, guessing bytes $S^6_5, \Delta S^6_5, S^0_4, S^3_4,$ $P_2, P_3, MK_2, MK_3, MK_6, MK_{12}, MK_{14}, MK_{15}$. The condition $\Delta S^3_8 = 0$ is a 8-bit condition, therefore if we index the table by $S^6_5, \Delta S^6_5, S^0_4, \Delta S^0_4, S^3_4, P_2, P_3, MK_2, MK_3, MK_6, MK_{12}, MK_{15}$ and take as outputs MK_{14}, for each set of inputs we get on average 1 valid output.	$2 \cdot (2^8)^{11} \cdot 2^7 C_B$
8	Build table T_4 by partial encryption, guessing bytes $S^4_6, \Delta S^4_6, S^5_6, \Delta S^5_6, MK_8$. The condition $\Delta S^3_7 = 0$ is a 8-bit condition, therefore if we index the table by $S^4_6, \Delta S^4_6, S^5_6, \Delta S^5_6$ and take as outputs MK_8, for each set of inputs we get on average 1 valid output.	$2 \cdot (2^8)^4 \cdot 2^7 C_B$
9	Build table T_5 by partial decryption, guessing bytes $C_6, \Delta C_6, C_7, \Delta C_7,$ $S^5_{29}, \Delta S^5_{29}, S^5_{28}, \Delta S^5_{28}, S^5_{27}, \Delta S^5_{27}, MK_0, MK_1, MK_5, MK_{10}, MK_{14}$. The condition $\Delta S^6_{26} = 0$ is a 8-bit condition, therefore if we index the table by $C_6, \Delta C_6, C_7, \Delta C_7, S^5_{29}, \Delta S^5_{29}, S^5_{28}, \Delta S^5_{28}, S^5_{27}, \Delta S^5_{27}, MK_0, MK_1, MK_{14}$ and take as outputs MK_5, MK_{10}, for each set of inputs we get on average 2^8 valid outputs.	$2 \cdot (2^8)^{13} \cdot 2^7 C_B$
10	Build table T_6 by partial decryption, guessing bytes $S^7_{26}, \Delta S^7_{26}, S^7_{27}, \Delta S^7_{27},$ $S^7_{28}, \Delta S^7_{28}, S^0_{29}, S^7_{29}, MK_0, MK_4, MK_9, MK_{13}$. The condition $\Delta S^0_{25} = 0$ is a 8-bit condition, therefore if we index the table by $S^7_{26}, \Delta S^7_{26}, S^7_{27}, \Delta S^7_{27}, S^7_{28}, \Delta S^7_{28}, S^0_{29}, S^7_{29}, MK_0, MK_4, MK_{13}$ and take as outputs MK_9, for each set of inputs we get on average 1 valid output.	$2 \cdot (2^8)^{11} \cdot 2^7 C_B$
11	Build table T_7 by partial decryption, guessing bytes $S^1_{25}, \Delta S^1_{25}, S^1_{26}, \Delta S^1_{26},$ $S^1_{27}, S^1_{28}, S^2_{28}, MK_7, MK_8, MK_{12}$. The conditions $\Delta S^2_{25} = 0$ and $\Delta S^2_{24} = 0$ are a 9-bit condition, therefore if we index the table by $S^1_{25}, \Delta S^1_{25}, S^1_{26}, \Delta S^1_{26}, S^1_{27}, S^1_{28}, S^2_{28}, MK_7, MK_8$ and take as outputs MK_{12}, for each set of inputs we get on average $\frac{1}{2}$ valid outputs.	$2 \cdot (2^8)^{10} \cdot 2^7 C_B$
12	Each evaluation of the first 3 steps corresponds to some evaluated bytes which will be the inputs of the hash tables, along with the outputs of the previous tables, therefore for each remaining pair and subkeys guesses for steps 1-3, the tables can be accessed and they produce some wrong subkeys (2^{31} in particular).	
13	The remaining keys are exhaustively tested.	